VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

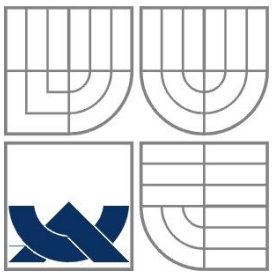
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

L-SYSTÉMY A JEJICH APLIKACE V POČÍTAČOVÉ GRAFICE

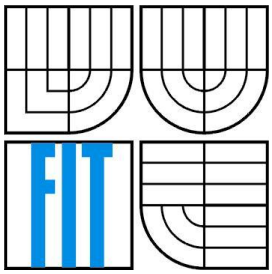
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ZDENĚK SOJMA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

L-SYSTÉMY A JEJICH APLIKACE V POČÍTAČOVÉ GRAFICE

L-SYSTEMS AND THEIR APPLICATIONS IN COMPUTER GRAPHICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ZDENĚK SOJMA
AUTHOR

VEDOUCÍ PRÁCE — ING. JIŘÍ KOUTNÝ
SUPERVISOR

BRNO 2009

Abstrakt

Bakalářská práce popisuje deterministické bezkontextové L-systémy (D0L-systémy), jejichž funkcí je paralelní přepisování symbolů v řetězci za účelem modelování rostlinám podobných struktur, především listů rostlin. Dále se zabývá využitím těchto L-systémů v počítačové grafice a vysvětluje využití stochastických L-systémů, které ovlivňují topologii a geometrii rostliny. Ty využívají náhodnosti při přepisu symbolů v L-systému, a tím nám dovolují modelovat celou řadu navzájem rozdílných, ale přesto podobných modelů rostlin. Interpret L-systémů je implementován v prostředí .NET s využitím knihovny OpenGL.

Abstract

This Bachelor's thesis describes deterministic context-free L-systems (D0L-systems), which fiction is parallel rewriting symbols in string for purpose of modeling plant-like structures, mainly plant leaves. Next it shows how to use such L-systems in computer graphics and explains usage of stochastic L-systems. They effect topology and geometry of plant by randomizing interpretation of the L-system and allow us to generate whole class of respectively different but still simile leaves. L-system interpreter is implemented in .NET framework with OpenGL library.

Klíčová slova

L-systém, OpenGL, modelování listů, stochastika, přepisování, želví grafika, fraktál, 0L.

Keywords

L-system, OpenGL, modeling leaves, stochastic, re-writing, turtle graphics, fractal, 0L.

Citace

Sojma Zdeněk: L-systémy a jejich aplikace v počítačové grafice, bakalářská práce, Brno, FIT VUT v Brně, 2009

L-systémy a jejich aplikace v počítačové grafice

Prohlášení

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

.....
Zdeněk Sojma
15. května 2009

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu Ing. Jiřímu Koutnému, který byl v práci opravdovou oporou a pomohl vždy, když bylo třeba.

Obsah

1 Úvod

V dnešní době se velkou rychlostí vyvíjí *moderní počítačová grafika*, která se již dávno rozvinula do samostatné vědní disciplíny a těší se velkému zájmu nejen programátorů, ale i běžných uživatelů. Dnes se využívá při tvorbě všemožných aplikací, které mají člověka na první pohled hlavně zaujmout a oslovit, a proto je velmi důležité, aby výsledný obraz vypadal pěkně a pokud možno reálně. Abychom splnili tyto požadavky, potřebujeme vytvořit modely různých předmětů, které vídáváme okolo sebe, a toho se dá dosáhnout v zásadě třemi způsoby. První způsob je takový, že můžeme nějaký reálný objekt nasnímat (například digitálním fotoaparátem nebo prostorovým scannerem). Tomuto postupu se říká *na obrazech založené modelování (image based modeling)* [19] a s nástupem kvalitních digitálních fotoaparátů se mu dostává stále větší obliby.

Pokud bychom ovšem chtěli vytvořit nějakou rozsáhlejší prostorovou scénu nebo objekt s velkými detaily, tak objem dat získaných ze snímání povrchů těchto vyobrazovaných předmětů by byl velmi velký a práce s nimi by byla téměř nemožná. Proto se na modelování podobných scén začalo využívat *analytického modelování,* neboli modelování, kdy animátor zadává tvar a vlastnosti objektu ručně. Toto modelování je velmi náročné na čas, talent a zkušenosti animátora, ale zase výsledný obraz je většinou velmi přesný a detailnější než u na obrazech založeného modelování, protože animátor přesně ví, nebo si umí představit, vnitřní strukturu objektu (například když jeden objekt zakrývá druhý), což takový scanner nedokáže.

Pokud ovšem budeme chtít modelovat rostliny a všeobecně přírodní útvary, tak narážíme na další nové problémy. Rostliny mají totiž z hlediska geometrie zajímavé vlastnosti, jako je *souměrnost listů* a *květů*, či *soběpodobnost* (viz Kapitola 2.1). Matematicky se tato vlastnost nazývá *invariance vůči změně měřítka*, což znamená, že část objektu je podobná celku při pohledu v různém přiblížení. *Soběpodobnost* můžeme vidět například u listu kapradiny, jehož část čepele je podobná čepeli jako celku (Obrázek 2.2), nebo když se podíváme z blízka na kámen, tak je podobný celé hoře. Nejen z tohoto důvodu je *analytické modelování* rostlin a dalších přírodních útvarů (ostrovů, mraků) velmi náročné, ale také protože i objem vytvářených dat je obrovský a vynaložená práce většinou neodpovídá nákladům, které s ní souvisejí. Tudíž se k modelování těchto útvarů začalo využívat *procedurálního modelování,* při němž animátor již nezadává přímo tvar objektu, ale spíše způsob nebo princip, jakým bude objekt vytvořen. Tento styl modelování má hlavní výhodu v tom, že objem analytických dat je poměrně malý a pouhou změnou počátečních podmínek se dá jednoduše objekt modifikovat. Při procedurálním modelování se mimo jiné využívá tzv. *fraktální geometrie* (viz kapitola 2), jejíž hlavní výhodou je paměťová nenáročnost a pro modelování rostlin, hor, mraků, kamenů neexistuje nám známý způsob s lepšími výsledky. Fraktální geometrie se dělí do několika kategorií:

- Dynamické systémy s fraktální strukturou
- L-systémy (Lindenmayerovy systémy)
- Stochastické fraktály
- Systémy iterovaných funkcí IFS

Důležitou vlastností fraktální geometrie z pohledu počítačové grafiky je, že každá z těchto kategorií se vytváří pomocí podobných algoritmů.

Tato práce se zabývá *L-systémy*, které jsou založené na přepisovacích gramatikách a jejichž využití se nachází nejen pro modelování rostlin, ale také například pro modelování říčních toků, ostrovů, pohoří, buněčných organismů, korálů nebo dokonce pro modelování budov. Moderní L-systémy spolu s pokročilou počítačovou grafikou umožňují vytvořit realistické modely přírody, přičemž využívají například i vliv gravitace, slunečního světla, vzdálenosti od vody nebo interakce s okolím.

1.1 Cíle práce

Cílem této práce je vytvořit aplikaci, která bude schopna demonstrovat možnosti *stochastických bezkontextových L-systémů* a jejich základních technik. Pomocí programu by mělo být možno názorně (graficky, didakticky, apod.) zobrazit techniku generování řetězce pomocí přepisovacích gramatik. Dále zobrazit výsledný model ve všech možných úhlech natočení tak, aby byl jasně patrný jeho vývoj a struktura. Model by měl být snadno porovnatelný s dalšími modely, vygenerovanými podle stejných stochastických pravidel, nebo s modely, které byly vygenerovány pomocí jednoduchých změn v pravidlech či parametrech daného L-systému. Aplikace je zaměřena na generování listů rostlin a zvládá vygenerovat modely o složitosti od nejjednodušších *deterministických lineárních fraktálů* až po komplikovanější *stochastické stromové 0L-systémy*.

Kapitola Fraktální geometrie obsahuje úvod do fraktální geometrie s vysvětlením jejích základních pojmů a principů. Kapitola L-systémy se zabývá podrobným popisem principu generování a interpretace modelů na základě přepisovacích technik a želví grafiky. V kapitole Modelování listů rostlin je obsažena hlavní část bakalářské práce, ve které jsou rozděleny techniky pro modelování většiny typů listů. Modelování stromů uvádí do problematiky interpretace kmenů rostlin. Šestá kapitola popisuje způsob vytvoření vlastní aplikace a návrhu kódu L-systému a v poslední kapitole je nastíněn další možný vývoj projektu, který je zaměřený především na rozšíření možností interpretace modelu a na další typy L-systémů.

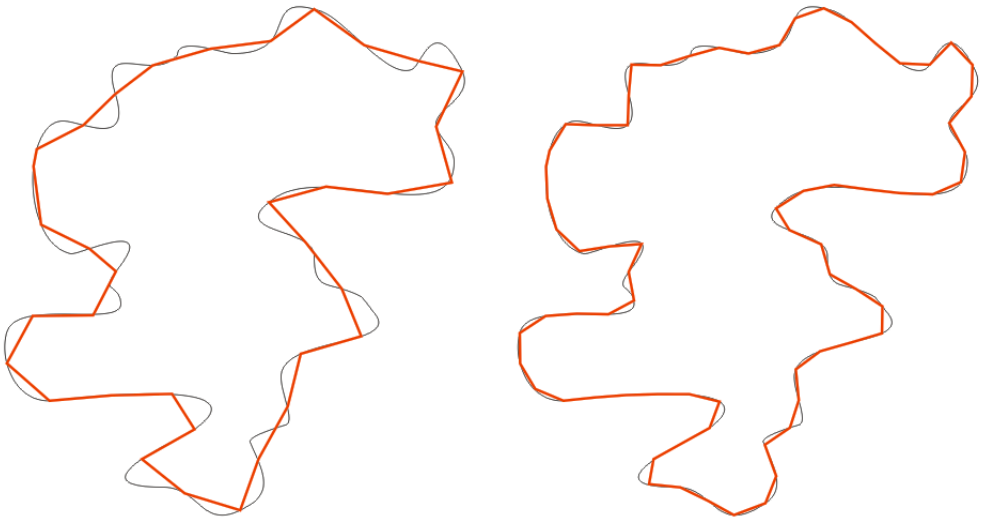
2 Fraktální geometrie

V této kapitole si popíšeme *fraktální geometrii*, protože právě touto technikou se v poslední době popisuje geometrický vzhled našeho světa. Dříve se k tomuto používala *Euklidovská geometrie*, která je schopna jednoduše popsat základní útvary jako koule, válec, trojúhelník, ale pro obrazce, jako je například již v kapitole 1 zmíněný list kapradiny (Obrázek 2.2), by se muselo napsat mnoho rozsáhlých nepřehledných rovnic, což u fraktální geometrie není nutné a místo nich se objekt popisuje pomocí algoritmů.

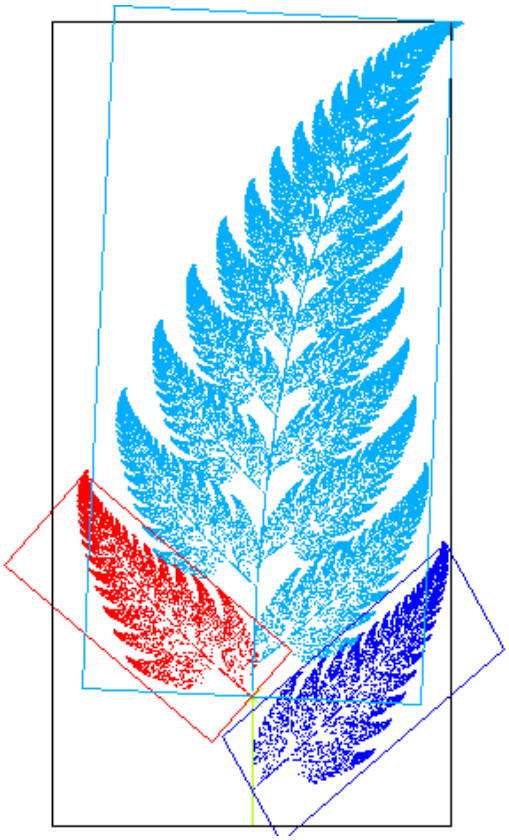
Jako první pojmenoval pojem *fraktál* polský vědec Benoin B. Maldebort [8] v šedesátých letech minulého století. Fraktály se vyskytují v živé i neživé přírodě a lze je nejlépe popsat jako geometrické objekty, jejichž motiv se opakuje až do *nekonečna* (soběpodobnost). V reálném světě ovšem vždy existují hranice, za které se nedá jít, takže pojem nekonečno je nutno brát spíše z matematického hlediska. Tuto fraktální strukturu můžeme například vidět nejen u stromů či keřů, kdy větvička je podobná celému porostu, ale také u mraků, kamenů, pobřeží ostrovů či břehů říčních toků.

Fraktály lze definovat jako *nekonečně členité útvary* [16]. Pokud budeme chtít změřit délku nějaké přírodní hranice okolo státu a budeme mít vyfotografovanou mapu s určitým měřítkem (např. 1:1 000 000), tak dokážeme například pomocí krokování kružítkem změřit délku této hranice. Ovšem když dostaneme do ruky jinou mapu s měřítkem řádově nižším (1: 10 000), tak nám pomocí stejného způsobu měření vyjde délka hranice delší. Z čehož vyplývá, že se změnou měřítka se změnila i délka výsledného obrazce, protože s podrobnější mapou jsme měřili po menších krocích a díky tomu jsme lépe kopírovali členitý přírodní terén (Obrázek 2.1). Dále pokud bychom obvod měřili třeba krokováním pěšky, tak by nám samozřejmě vyšlo ještě větší číslo, a pokud bychom šli až do subatomárního měření, kde by se nám délka kroku blížila limitě k nule, tak by obvod ostrova rostl až do nekonečna. Z toho vyplívá, že stát o *konečném obsahu* má *nekonečnou délku hranice* [16]. Pokud ovšem půjdeme po této hranici pěšky, tak v reálném čase dojdeme na místo, ze kterého jsme vyšli. Tudíž je tedy nutné měřit tyto vzdálenosti po pevně daných krocích.

Fraktální geometrie se hojně využívá také například pro generování textur, kdy data takto vygenerované textury mají velikost několika desítek bitů (vektorový obrázek) oproti bitmapám, kde je tato velikost v řádech stovek kilobitů. Dále je možné využít fraktálů při animacích, kdy se do systému přidá další rozměr, který se bude považovat za čas. Této metody se využívá při zobrazování pohybujících se mraků či ohně.



Obrázek 2.1 Hranice státu změřené různým krokováním.



Obrázek 2.2 Soběpodobný list kapradiny převzatý z [3].

2.1 Soběpodobnost

Soběpodobnost (*self-similary*) je charakteristickou vlastností fraktálních objektů a je nazývaná také jako *soběpříbuznost* či matematicky *invariance vůči změně měřítka*. Značí se tím, že se část objektu podobá celku (nemusí být zcela úplně stejná) při pohledu v různém zvětšení. Soběpodobnou strukturu je možné rozdělit na struktury, které jsou zmenšenou kopií originálu (viz Obrázek 2.2).

Podle knihy [19] se soběpodobnost rozlišuje na *soběpodobnost přesnou* a *statickou*, jež jsou matematicky definované následovně (celý zbytek kapitoly vychází právě z této knihy):

Definice 1 Množina A je *přesně soběpodobná*, pokud je sjednocením konečného počtu transformovaných kopií sebe samé.

$$A = \bigcup_{i=1}^{n} \varphi_i(A) \qquad (2.1)$$

V tomto vztahu jsou transformace φ_i posunutí a rotace a každá z nich je zároveň změnou měřítka s koeficientem $S_i \in \langle 0,1 \rangle$, nebo jsou všechny tzv. průměrně kontraktivní. Pokud by součet koeficientů S_i přesáhl hodnotu jedné, množina by prostorově divergovala do nekonečna. Podmínka průměrné kontraktivity množiny transformací $\{\varphi_i, i = 1,2,\ldots,n\}$ má tvar:

$$0 < \sum_{i=1}^{n} S_i < 1 \qquad (2.2)$$

Přesně soběpodobná je například Kochova vločka (viz Obrázek 3.1).

Definice 2 Transformace $\varphi : U \rightarrow U$ je lineární, pokud $\varphi(r_1A + r_2B) = r_1\varphi(A) + r_2\varphi(B)$ pro všechna $A,B \in U$ a $r1,r2 \in \mathbb{R}$. U je vektorový prostor a $\mathbb{R}$ je množina reálných čísel.

Definice 3 Množina A je *statisticky soběpodobná,* pokud je sjednocením konečného počtu zmenšených kopií sebe samé, podle vztahu (2.1) a každá z kopií $\varphi_i(A)$ má stejné statistické charakteristiky, jako množina A. Říkáme, že $\varphi_i(A)$ a A jsou statisticky nerozlišitelné. Transformace φ_i musí být zároveň změnou měřítka s koeficientem $s_i \in (0,1)$. Jsou-li aplikované transformace lineární, resp. nelineární, je soběpodobná množina A lineární, resp. nelineární. Za zachování podmínky statistické soběpodobnosti v praxi obyčejně považujeme shodu směrodatné odchylky a průměru a ne všech statistických momentů.

Příkladem statistické soběpodobnosti je kámen a hora. Pokud budeme porovnávat vhodně vybranou fotografii kamene a hory, bude pro nás obtížné rozhodnout, co je horou a co kamenem. Jiným příkladem je nahrávka šumu z rádia na frekvenci, kde není žádná stanice. Pokud takový šum nahrajeme, například na magnetofonový pásek, a budeme ho přehrávat libovolnou rychlostí, bude znít s největší pravděpodobností stejně. Změna rychlosti přehrávání odpovídá změně měřítka.

3 L-systémy

Autorem *L-systémů* je maďarský biolog Aristid Lindenmayer, který je v roce 1968 představil jako *matematický teoretický rámec pro vývoj rostlinných organismů* [11]. Vycházejí ze systémů *paralelního přepisování řetězců* podle určitých pravidel, kde po několika *opakováních přepsání* (*derivacích*) se výsledný řetězec interpretuje graficky a to tak, že k některým symbolům výsledného řetězce je přiřazen jistý *geometrický význam* – například vykreslení či transformace objektu.

Formální definice L-systémů je popsána podle [11] následovně:

Definice 4 L-systémy jsou definované jako trojice $H = (V,P,\omega)$, kde V je konečná neprázdná množina symbolů. P je konečný počet pravidel typu $a \rightarrow x$, kde $a \in V$ a $x \in V^*$. A $\omega \in V^+$ je počáteční řetězec (*axiom*).

3.1 Přepisovací systémy

Přepisování řetězce (*derivace*) se provádí opakovaně po několik předem zadaných kroků, jak již bylo napsáno výše. Každý symbol v řetězci se paralelně nahradí řetězcem symbolů podle pravé strany zadaného pravidla, pokud se k danému symbolu žádné pravidlo nevztahuje, symbol se nepřepíše a v řetězci zůstává. Například pravidlo $a \rightarrow ab$ znamená nahrazení všech výskytů symbolů a v řetězci dvojicí symbolů ab (viz Obrázek 3.3, 2. derivační krok).

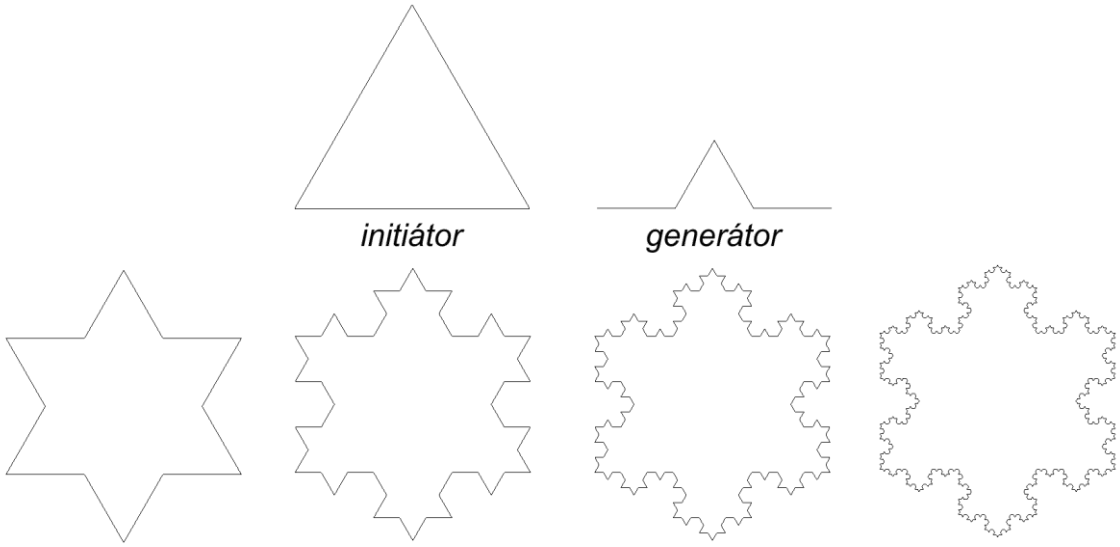
Asi nejznámějším příkladem L-systému je Kochova sněhová vločka, pojmenovaná podle jejího autora Helge von Kocha, kterou navrhl v roce 1905. Obrázek 3.1 vyobrazuje konstrukci této vločky postupně od první až po čtvrtou derivaci, jež byla popsána polským vědcem Benoidem B. Mandelbrotem následovně [8]:

> Vločka se skládá ze dvou obrazců – *iniciátoru* (*axiomu*) a *generátoru* (*pravidla*). Generátor je *orientovaná lomená čára*, která je složená z N stejných úseků délky r. Každá část konstrukce začíná lomenou čárou, jejíž každá přímá čára je nahrazena kopií generátoru, který je zmenšen a vyobrazen tak, aby měl stejné koncové body jako přímá čára iniciátoru, kterou nahradil.

Na začátku minulého století norský matematik Axel Thue [13] napsal první formální definici přepisovacího systému, která ovšem nebyla ještě příliš obsáhlá, a proto v padesátých letech minulého století přišel americký lingvista Avram Noam Chomsky s daleko vypracovanější definicí, kterou využil při popisu syntaktických rysů přirozených jazyků ve své práci s formálními gramatikami [6].

O pár let později pánové John Backus a Peter Naur představili takzvanou Backus-Naurovu Formu (BNF), která se používá k vyjádření bezkontextových gramatik a kterou využili ve svém programovacím jazyce ANGOL-60 [2]. Brzy na to byla rozpoznána ekvivalence mezi

Backus-Naurovou formou a Chomského bezkontextovou gramatikou a vznikl nový vědní obor založený na přepisování sady řetězců nazývaných formální gramatiky.



Obrázek 3.1 Konstrukce Kochovy sněhové vločky.

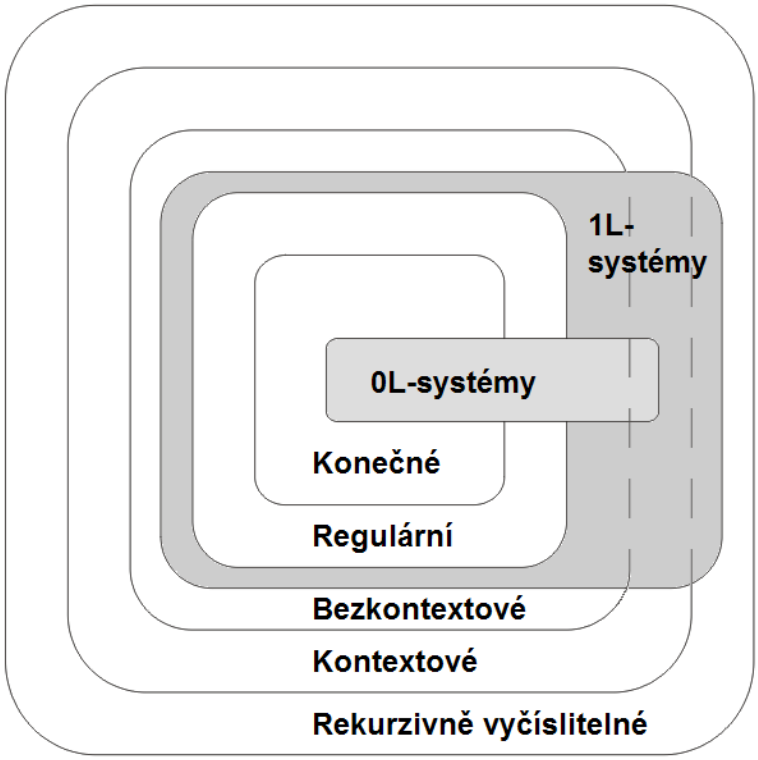
V roce 1968 přišel Aristid Lindenmayer s novým typem přepisovací gramatiky, který nazval *L-systémem*. Zásadní rozdíl mezi již v tu dobu známou Chomského gramatikou je v tom, že k přepisování dochází paralelně pro všechny symboly v řetězci a ne jak tomu bylo donedávna, kdy se přepisování aplikovalo postupně symbol po symbolu. Díky tomuto, pro někoho zcela nepatrnému rozdílu, se dají generovat úplně odlišné obrazce, což nasměrovalo L-systémy spíše k *biologickému zaměření*, protože se dá například simulovat buněčné dělení, které probíhá ve všech buňkách systému paralelně. Tento fakt má vliv na formální vlastnosti L-systému oproti Chomského gramatikám (viz Obrázek 3.2).

3.2 D0L-systémy

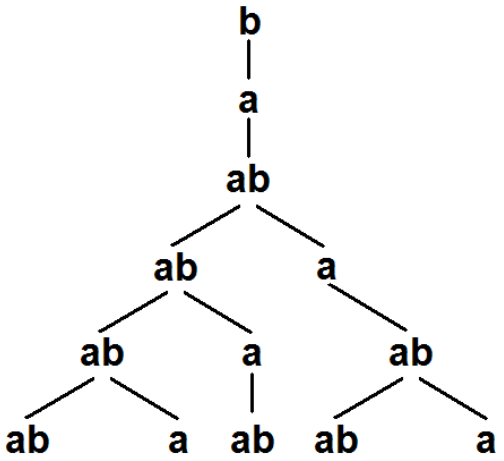
Nejjednodušším typem L-systému jsou *deterministické bezkontextové L-systémy*, které mají zkratku *D0L-systémy*. Nula v názvu znamená, že se jedná o *bezkontextový přepis* – pravidlo se na symbol aplikuje vždy a symboly vedle něj (kontext) se neberou v úvahu, na rozdíl od *1L* a *2L-systémů*, kde se symbol přepíše pouze právě tehdy, když je symbol v kontextu s ostatními podle zadaného pravidla. 1L-systémy jsou omezeny vždy pouze na jednu stranu symbolu, 2L na obě dvě. Písmeno *D* značí *determinismus* – pro každý symbol v řetězci existuje maximálně jedno pravidlo typu $a \rightarrow x$. Pokud poslední pravidlo omezíme tak, že ke každému symbolu v řetězci existuje právě jedno pravidlo typu $a \rightarrow x$, tak získáme tzv. *P0L-systém*, neboli systém, který neobsahuje žádná ε *pravidla* [15].

Mějme pravidla p_1: $b \rightarrow a$, p_2: $a \rightarrow ab$ a počáteční řetězec zvaný *axiom* $\omega=b$, kterým začíná proces přepisování. V prvním derivačním kroku bude symbol b nahrazen symbolem a podle pravidla $b \rightarrow a$. V dalším bude symbol a nahrazen řetězcem symbolů ab podle pravidla

$a \rightarrow ab$. Ve třetím derivačním kroku bude nahrazen opět symbol *a* řetězcem *ab* a paralelně s ním i symbol *b* symbolem *a* podle stejných pravidel. Vznikne tedy řetězec *aba* a můžeme pokračovat stejným postupem dál, až se například v pátém derivačním kroku dostaneme k řetězci *abaababa*. Obrázek 3.3 podrobněji znázorňuje tento postup.



Obrázek 3.2 Vztah mezi Chomského gramatikami a Lindenmayerovými systémy.



Obrázek 3.3 Příklad 5ti derivačních kroků D0L-systému.

3.2.1 Formální definice D0L-systému

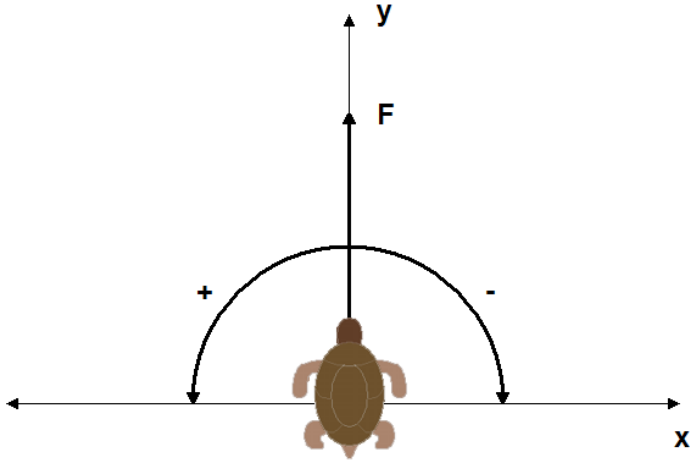
V této kapitole je napsána formální definice D0L-systému převzatá z [12].

Definice 5 Nechť V je abeceda symbolů, tak V^* je množina všech řetězců nad abecedou V a V^+ je množinou všech neprázdných řetězců nad V. Pak *0L-systém* je definován uspořádanou trojicí $G = (V,\omega,P)$, kde V je abeceda symbolů, $\omega \in V^+$ je neprázdný řetězec symbolů zvaný *axiom* a $P \subset V \times V^*$ je konečná množina přepisovacích pravidel. Pravidlo $(a, \chi) \in P$ je napsáno jako $a \rightarrow x$. Symbol a nazýváme *předchůdcem* (*predecessor*) a řetězec χ *následníkem* (*successor*) tohoto pravidla. Předpokládá se, že pro každý symbol $a \in V$ existuje alespoň jeden řetězec $\chi \in V^*$ stylu $a \rightarrow \chi$. Jestliže pro předchůdce $a \in V$ není specifikováno žádné pravidlo, předpokládá se, že je přidáno identické pravidlo stylu $a \rightarrow a$ do P. *0L-systém* je *deterministický* (*D0L-systém*) právě tehdy, když pro každý symbol $a \in V$ existuje maximálně jedno pravidlo $\chi \in V^*$ stylu $a \rightarrow \chi$.

Definice 6 Nechť $\mu = a_1 \ldots a_m$ je libovolný řetězec nad V. Pak řetězec symbolů $\nu = \chi_1 \ldots \chi_m \in V^*$ je přímo *derivován* (*generován*) pomocí μ, tak zapisujeme $\mu \Rightarrow \nu$ právě tehdy, když $a_i \rightarrow \chi_i$ pro všechna $i = 1,\ldots,m$. Řetězec ν je generován z G derivací *délky n*, jestliže existuje vývojová sekvence řetězců $\mu_0,\mu_1,\ldots,\mu_n$ taková, že $\mu_0 = \omega$, $\mu_1 = \nu$ a $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$.

3.3 Želví grafika

Pro interpretaci modelů vytvořených pomocí L-systémů se využívá tzv. *želví grafiky* (*turtle graphics*). Želva je objekt, který je považován za imaginární kreslící zařízení. Je definovaná trojicí $H=(x,y,\alpha)$, kde x, y označuje pozici želvy v kartézském souřadném systému a α značí směr, kterým je želva natočená (viz Obrázek 3.4).



Obrázek 3.4 Orientace želvy v kartézském souřadném systému na počáteční pozici.

Pojem *želví grafika* vychází z programovacího jazyka *LOGO*, jehož princip modelování se využívá i v L-systémech. Podle literatury [16] písmeno *L* v názvu L-systémů vychází právě ze zkráceniny anglického sousloví *LOGO-like turtle*.

LOGO je jednoduchý funkcionální programovací jazyk, který vychází z programovacího jazyka *LISP* (syntaxe je však LISPu docela vzdálená) a byl vytvořen za účelem podpory *konstruktivního učení*. Hlavní postavou jazyka je právě pomyslná *želva* (*turtle*) pohybující se po písku na pláži. Když při svém pohybu má skloněný ocásek, tak za sebou v písku nechává čáru (kreslí úsečku), a když ocásek zvedne, tak se pohybuje bez toho, aby cokoli za ní zbylo (nic nekreslí) [17]. Tento princip je použit právě v L-systémech a dají se jím tak kreslit jednoduché obrazce složené z úseček.

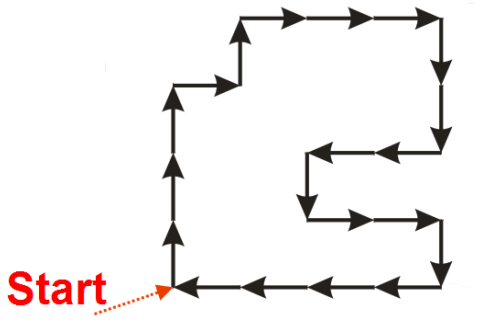
3.3.1 Interpretace řetězce želvou

Želva se pohybuje podle symbolů ve výsledném řetězci, který je po několika derivačních krocích vytvořený pomocí pravidel L-systému. Každý symbol v řetězci má předem určený význam a ovlivňuje chování želvy. Například když se želva nachází v bodě *a* a má právě interpretovat symbol *F*, tak se posune vpřed o vzdálenost *d* do bodu *b* a vykreslí tak úsečku *ab*.

Základní symboly interpretované želvou (Obrázek 3.4):

- `F` : posunutí želvy vpřed o délku *d* z původních souřadnic (x, y, α) na $(x+d*\cos\alpha, y+d*\sin\alpha, \alpha)$ a vykreslení úsečky mezi těmito body.
- `f` : posunutí želvy vpřed o délku *d* z původních souřadnic (x, y, α) na $(x+d*\cos\alpha, y+d*\sin\alpha, \alpha)$ bez kreslení úsečky mezi těmito body.
- `+` : rotace želvy doleva o úhel δ, neboli změna pozice želvy na $(x, y, \alpha+\delta)$.
- `-` : rotace želvy doprava o úhel δ, neboli změna pozice želvy na $(x, y, \alpha-\delta)$.

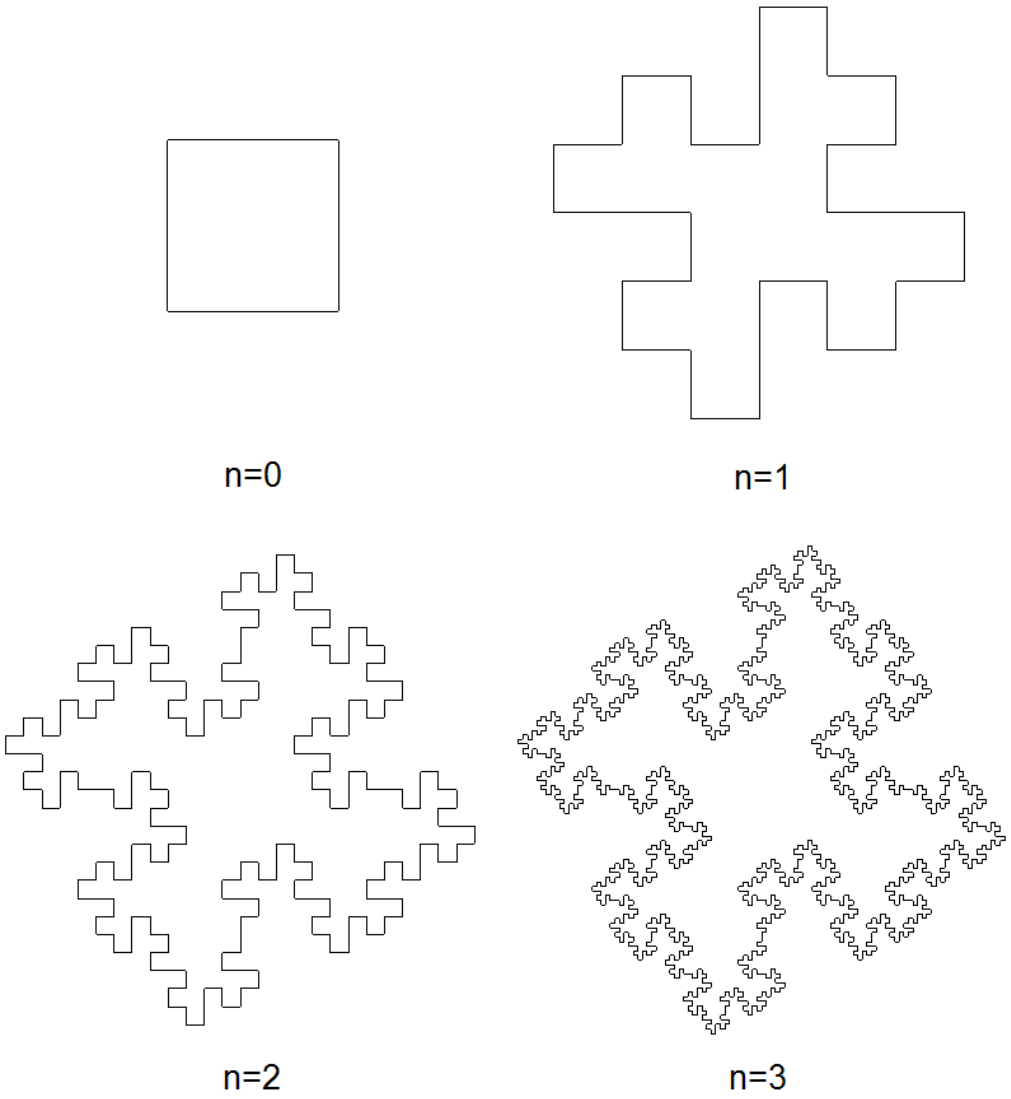
Máme-li interpretovat řetězec ν, tak musíme mít zadán počáteční stav želvy (x_0, y_0, α_0) a dvě základní konstanty – *d (délka kroku)* a δ *(úhel natočení)*. Poté se řetězec ν projde od prvního symbolu k poslednímu, a když se najde symbol, který má pro interpretaci obrazce nějaký význam, tak se provede akce s ním související a pokračuje se dále v průchodu řetězce. Výsledný řetězec je obrazec složený z úseček. Obrázek 3.5 ukazuje vykreslení daného řetězce pomocí želví grafiky.



`FFF-F+F-FFF-FF-FF+F+FF-F-FFFF`

Obrázek 3.5 Želví interpretace řetězce.

Pěkným příkladem D0L-systému vytvořeného želví grafikou je Kochův ostrov [8], jenž je vyobrazen na následujícím obrázku a je tvořen pomocí pod ním vypsaných pravidel, kde `n` značí počet derivačních kroků.

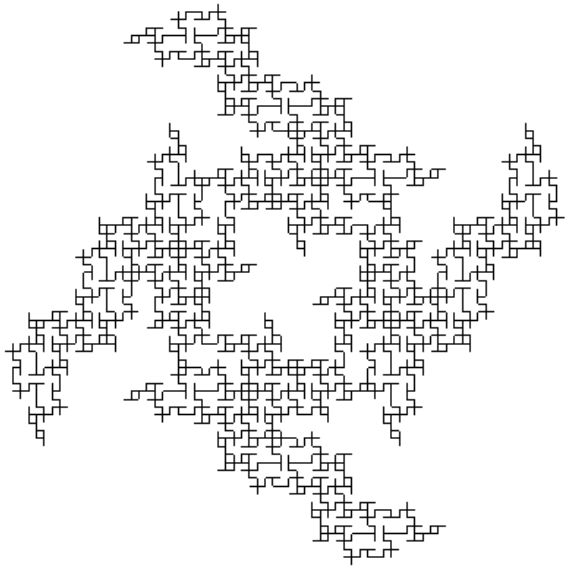


`n = 3, δ = 90°`
`ω :` `F-F-F-F`
`p :` `F → F-F+F+FF-F-F+F`

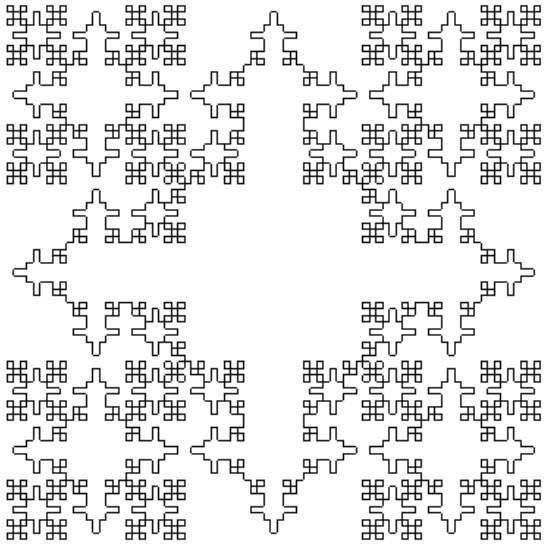
Obrázek 3.6 Interpretace Kochova ostrova od nulté po třetí derivaci.

Obrázky nejsou vykresleny vůči sobě v poměru 1:1. Je u nich postupně zmenšováno měřítko, protože při každé derivaci se délka všech hran zvětší 4x (je zadána pevná délka kroku). Pokud bychom chtěli, aby nám velikost ostrova nerostla, tak bychom mohli využít například *parametrických L-systémů*, u nichž se délka kroku dá libovolně měnit a které si popíšeme v kapitole 3.6.

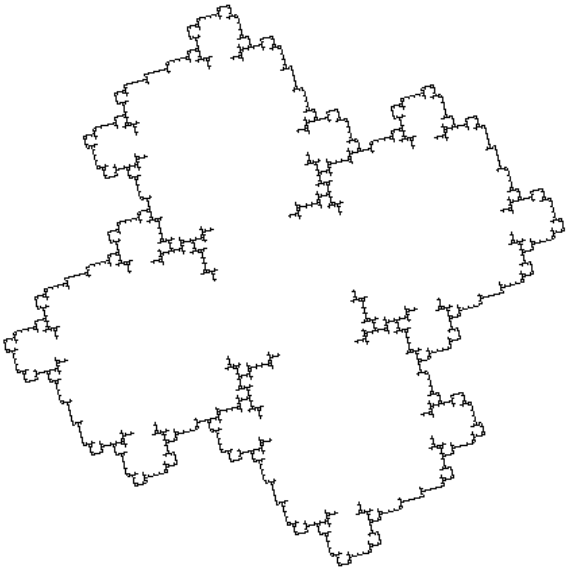
Na následujícím obrázku jsou vykreslené obrazce vytvořené D0L-systémem, u kterého jako předloha posloužil právě Kochův ostrov a byl v nich modifikován řetězec *následníka* (pravé strany pravidla). Obrazce vznikly mým experimentováním.



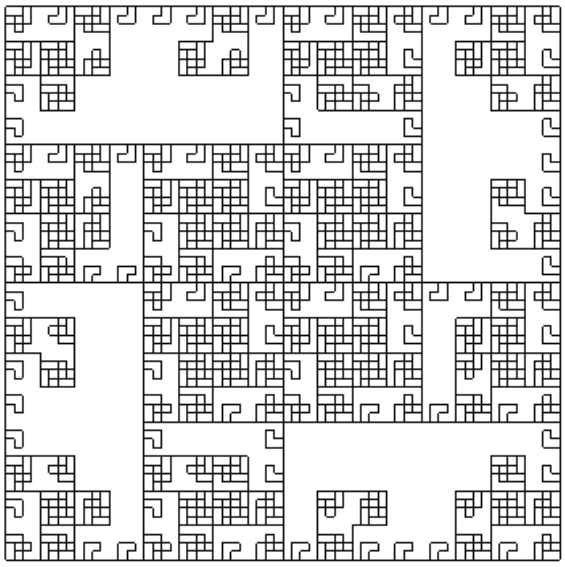
n = 4, δ = 90°
ω: F-F-F-F
p: F → FF+F-FF--F+F



n = 4, δ = 90°
ω: F-F-F-F
p: F → F-FF+F+F-F



n = 4, δ = 90°
ω: F-F-F-F
p: F → FF-F--FF-FF

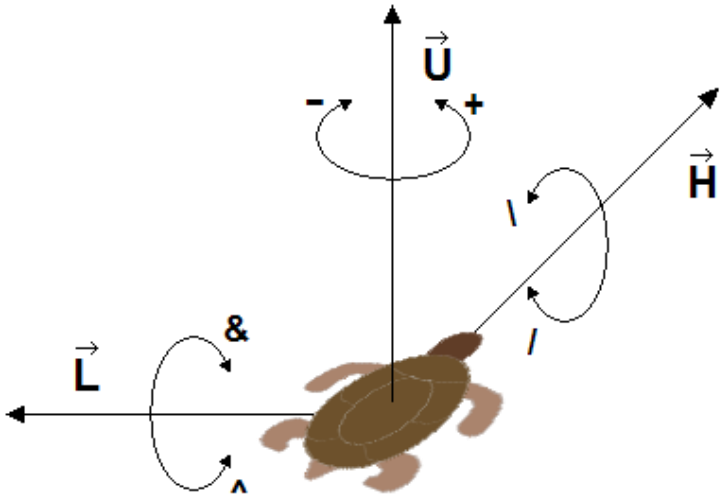


n = 4, δ = 90°
ω: F-F-F-F
p: F → FFF-FF-FF-F-F+F-FF

Obrázek 3.7 L-systémy vycházející z Kochova ostrova s modifikovaným následníkem.

3.3.2 Interpretace želvou v 3D prostoru

Pro interpretaci modelů v třídimenzionálním prostoru musíme začít pozici želvy popisovat jako trojici *(x, y, z)* a soustavu tří vektorů $\vec{H}$, $\vec{L}$, $\vec{U}$, které znázorňují natočení želvy oproti souřadným osám. Tato změna je důležitá, jelikož v 3D prostoru se želva otáčí již kolem tří os, a ne pouze kolem jedné, jak tomu bylo v rovině. S použitím soustavy vektorů poprvé přišli pánové Albeson a diSessa [1]. V této trojici vektor $\vec{H}$ značí směr, kterým se želva dívá (*heading*), vektor $\vec{L}$ směr prostoru vlevo od želvy (*left*) a vektor $\vec{U}$ směr nahoru nad želvou (*up*), (viz Obrázek 3.8). Vektory mají stejnou (nejlépe jednotkovou) délku a jsou na sebe navzájem kolmé.



Obrázek 3.8 Orientace želvy v 3D prostoru.

Pro vektory platí vzájemný vztah $\vec{H} \times \vec{L} = \vec{U}$ a rotace želvy se vypočítává podle rovnice

$$[\ \vec{H}'\ \vec{L}'\ \vec{U}'\ ] = [\ \vec{H}\ \ \vec{L}\ \ \vec{U}\ ] * \mathbf{R}$$

kde ***R*** značí jednu ze tří rotačních matic. Matici vybíráme podle osy okolo, které se želva otáčí.

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

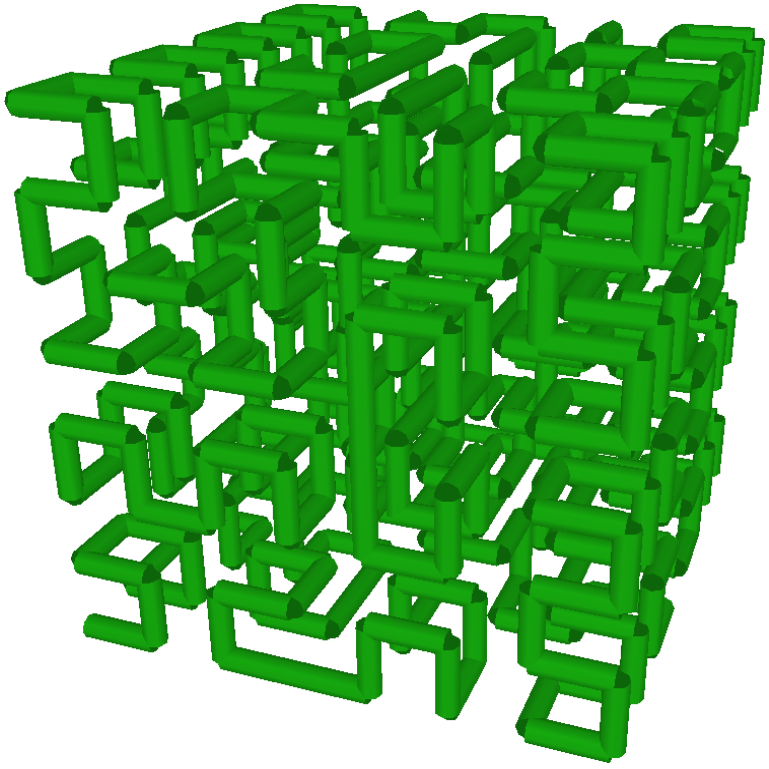
$$R_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

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

Pomocí následujících symbolů želva rotuje v prostoru (symboly pro pohyb vpřed zůstávají stejné):

- `+` : rotace želvy o úhel δ s použitím rotační matice $R_U(\delta)$.
- `−` : rotace želvy o úhel $-\delta$ s použitím rotační matice $R_U(-\delta)$.
- `|` : rotace želvy o $180°$ s použitím rotační matice $R_U(180°)$.
- `&` : rotace želvy o úhel δ s použitím rotační matice $R_L(\delta)$.
- `∧` : rotace želvy o úhel $-\delta$ s použitím rotační matice $R_L(-\delta)$.
- `\` : rotace želvy o úhel δ s použitím rotační matice $R_H(\delta)$.
- `/` : rotace želvy o úhel $-\delta$ s použitím rotační matice $R_H(-\delta)$.

Asi nejnázornějším příkladem třírozměrného obrazce vytvořeného pomocí D0L-systému je třídimenzionální Hilbertova křivka [14] (Obrázek 3.9).



n = 3, δ = 90°

ω: A

p_1: A → B−F+CFC+F−D&F∧D−F+&&CFC+F+B//

p_2: B → A&F∧CFB∧F∧D∧∧−F−D∧|F∧B|FC∧F∧A//

p_3: C → |D∧|F∧B−F+C∧F∧A&&FA&F∧C+F+B∧F∧D//

p_4: D → |CFB−F+B|FA&F∧A&&FB−F+B|FC//

Obrázek 3.9 Trojrozměrná Hilbertova křivka.

Jako příklad třírozměrného objektu, který je již biologického charakteru, může být například keř, jež je vyobrazen na dalším obrázku. K jeho generování byly využity techniky, jenž budou představeny v následujících kapitolách 3.5, 4 a 5 (větvení segmentů, generování ploch, tloušťka segmentu).

n = 7, δ = 22.5°
ω: A
p_1: A → [&FL!A]/////[&FL!A]///////[&FL!A]
p_2: F → S/////F
p_3: S → FL
p_4: L → [^^{-f+f+f-|-f+f+f}]

Obrázek 3.10 Příklad trojrozměrného keře vygenerovaného D0L-systémem.

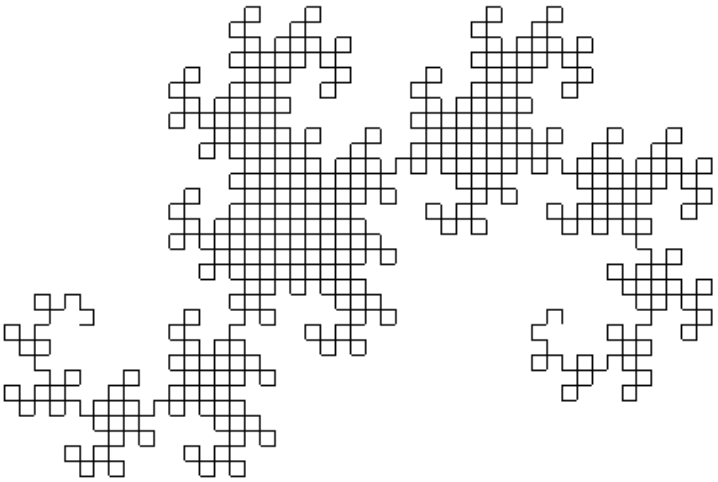
3.4 Přepisování řetězců

Pro přepisování řetězců se využívá dvou metod, které jsou od sebe navzájem principiálně odlišné, ale přesto se jimi dají vytvářet ekvivalentní modely. Rozhodnutí, kterou metodu vybrat tedy závisí pouze na autorovi L-systému, ale většinou se volí ta, která je v daný moment přehlednější.

Metody se dělí na *metodu přepisování hran (edge rewriting)* a *metodu přepisování uzlů (node rewriting)* [11]. Hlavním rozdílem těchto metod je, že při přepisování hran se nahrazují hrany obrazce, neboli symboly, pomocí kterých želva kreslí objekty (nám zatím známé symboly

f a *F*) a při přepisování uzlů jeho vrcholy, neboli symboly, které nemají žádný geometrický význam (neinterpretují se). Kvůli přepisování vrcholů se tedy do L-systému musí zavést nové symboly, které lze použít na levé straně pravidla, a přitom na interpretaci modelu z výsledného řetězce nemají žádný vliv.

Na následujícím obrázku je znázorněna dračí křivka, která je vygenerována jak pomocí přepisování hran (a), tak zároveň i podle přepisování uzlů (b).



a.) $n = 10,\ \delta = 90°$

ω: F_l

p_1: $F_l \rightarrow F_l+F_r+$

p_2: $F_r \rightarrow -F_l-F_r$

b.) $n = 10,\ \delta = 90°$

ω: Fl

p_1: $l \rightarrow l+rF+$

p_2: $r \rightarrow -Fl-r$

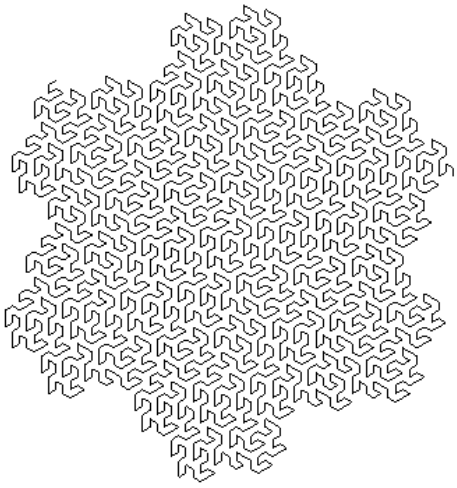
Obrázek 3.11 Dračí křivka vytvořená pomocí přepisování hran (a) a přepisování uzlů (b).

3.4.1 FASS křivky

Pomocí obou přepisovacích technik se vytváří další zajímavé křivky, jako jsou například *FASS* křivky (Obrázek 3.12). Zkratka FASS (*space-filling, self-avoiding, simple and self-fimilar*) znamená, že se jedná o tzv. *prostor vyplňující, neprotínající se, jednoduchou* a *soběpodobnou* křivku. Algoritmus na konstrukci těchto křivek pomocí přepisování hran vymyslel McKeen [10]. Princip je takový, že při přepisu symbolu F_l se vyplní volný prostor vlevo od přepsané úsečky a při přepisu F_r ten vpravo. Následující derivační kroky opět vyplní prostor vpravo nebo vlevo od hrany a vznikají tak složitější FASS křivky. Obrázek 3.13 vyobrazuje princip tohoto vyplňování na prvních dvou derivacích McKeenovy e-křivky, kde nahoře vyplňování začíná od pravé strany (*axiom* F_r) a dole od levé strany hrany obrazce (*axiom* F_l).

Další modely vygenerované odlišným typem přepisování jsou zobrazeny v kapitole 3.5.1 (přepisování hran viz Obrázek 3.16 nahoře a přepisování uzlů dole). Dělení na dva typy přepisování je důležité obzvlášť pro generování křivek vyplňujících prostor, které se navzájem

neprotínají, což je motivováno přírodou, kde se rostlinám také navzájem neprotínají větvičky, listy a podobně.



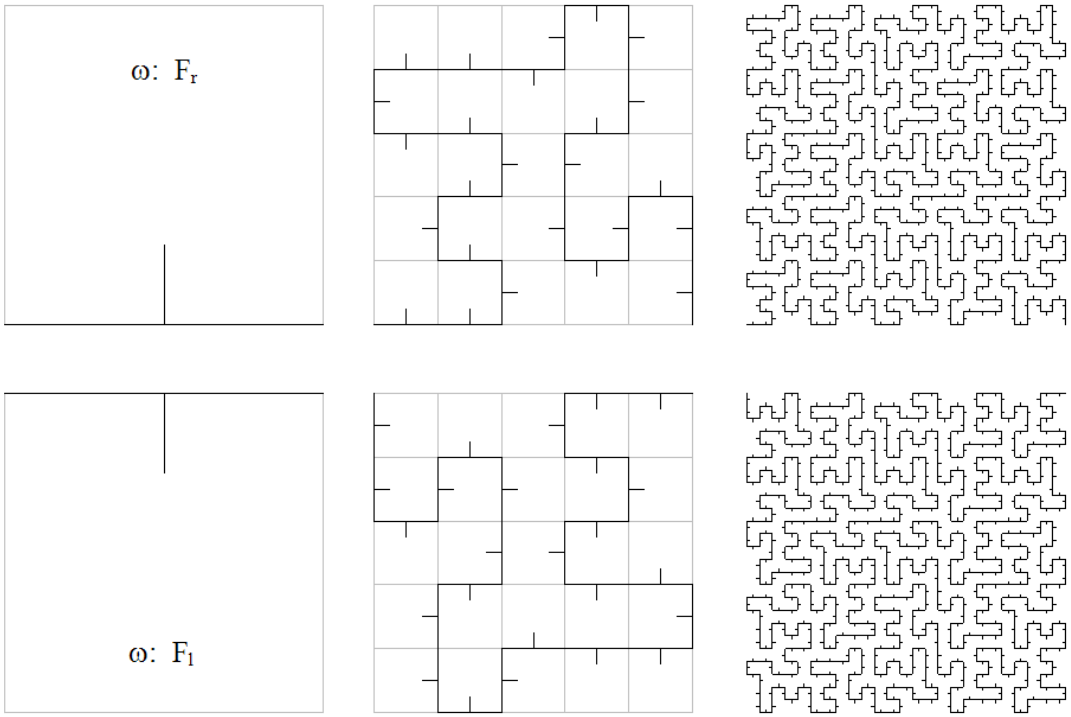
$n = 4,\ \delta = 60°$

$\omega:\quad F_l$

$p_1:\quad F_l \rightarrow F_l+F_r++F_r-F_l--F_lF_l-F_r+$

$p_2:\quad F_r \rightarrow -F_l+F_rF_r++F_r+F_l--F_l-F_r$

Obrázek 3.12 Gosperova FASS křivka.



$p_1: F_l \rightarrow F_lF_l+F_r+F_r-F_l-F_l+F_r+F_rF_l-F_r-F_lF_lF_r+F_l-F_r-F_lF_l-F_r+F_lF_r+F_r+F_l-F_l-F_rF_r+$

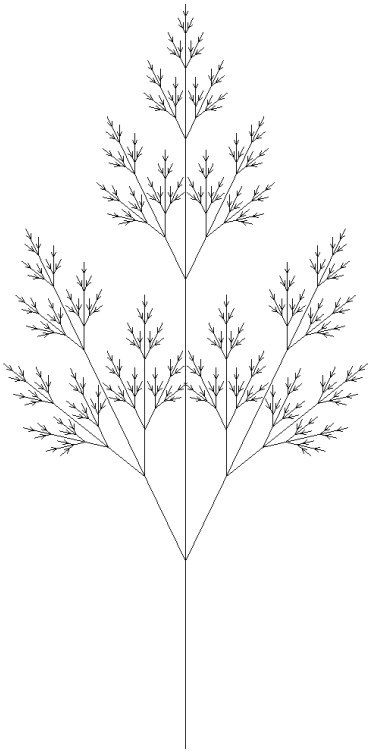
$p_2: F_r \rightarrow -F_lF_l+F_r+F_r-F_l-F_lF_r-F_l+F_rF_r+F_l+F_r-F_lF_rF_r+F_l+F_rF_l-F_l-F_r+F_r-F_l-F_l-F_rF_r$

Obrázek 3.13 Konstrikce tzv. e-křivky vytvořené McKeenem.

3.5 Rozvětvující se struktury

Pomocí L-systémů, které byly popsány v předchozích kapitolách, nemůžeme generovat větvící se struktury, ale jen obrazce, složené pouze z na sebe navazujících se úseček určitých délek. Tyto obrazce sice vypadaly pěkně, ale pro generování rostlinných organismů, kterými se budeme dále zabývat, nám jejich možnosti nepostačují.

Rozvětvení u nich nebylo možné, protože pro želvu neexistoval způsob jak si zapamatovat svoji minulou pozici, možnost vrátit se k ní a začít tak generovat nový segment obrazce. Mohli bychom se sice pomocí symbolů v pravidle vrátit stejnou cestou zpátky, ale to by bylo mnohdy velmi komplikované, či téměř nemožné.



Obrázek 3.14 Větvící struktura vytvořená pomocí D0L-systému.

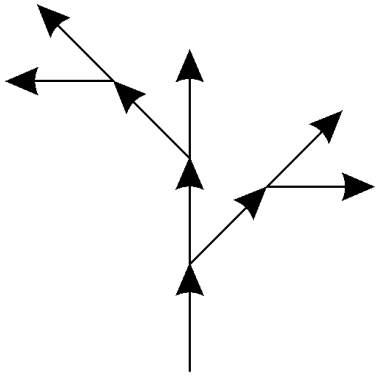
K vyřešení tohoto problému pomohlo zavedení takzvaných závorkových L-systémů, které danou paměť obsahují ve formě *zásobníku*, do kterého se ukládá a z něhož se načítá pozice i natočení želvy. Zásobník je typu *LIFO* (*Last In First Out*), neboli typu, kdy se pozice uložená na zásobníku jako poslední, vyjme ze zásobníku jako první. Výhoda řešení zásobníkem je v tom, že je docela jednoduché a dobře pochopitelné.

Želví zásobník obsahuje dvě klasické operace: uložení stavu želvy na zásobník (*push*) a načtení pozice želvy ze zásobníku (*pop*). Této funkčnosti využívají *závorkové L-systémy* (*bracketed L-systems*) popsané v následující kapitole.

3.5.1 Závorkové L-systémy

Abychom mohli využít výhody *zásobníku* v L-systémech, musíme definovat symboly, pomocí kterých želvě řekneme, že má svoji pozici uložit nebo načíst:

- `[` : uloží aktuální pozici želvy *(x, y, z)* na vrchol zásobníku. Informace na zásobníku obsahuje pozici želvy i úhel jejího natočení vůči osám ($\vec{H}$, $\vec{L}$, $\vec{U}$). Dále obsahuje všechny další možné informace, které tvůrce L-systému může potřebovat (např. barvu segmentu či šířku segmentu).
- `]` : načtení pozice želvy z vrcholu zásobníku a umístění želvy do dané pozice. Při této změně se nevykreslí žádná čára a jsou načteny i všechny další parametry, které si tvůrce uložil.

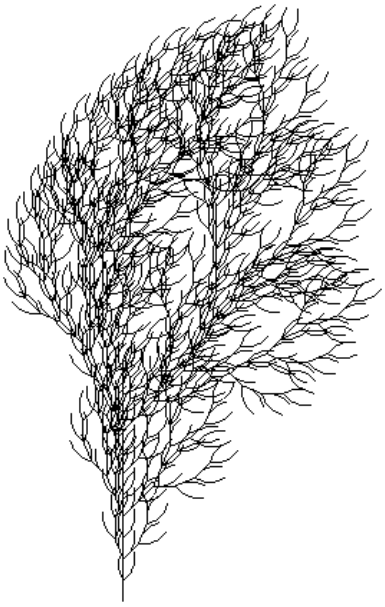


`F[-F[-F]F]F[+F[+F]F]F`

Obrázek 3.15 Příklad použití závorkových L-systémů.

V kapitolách 3.3.1 a 3.3.2 jsme si představili symboly, při jejichž interpretaci želva vykonává nějakou předdefinovanou akci (rotaci, posun, atd.). Tyto symboly se označují jako *nonterminální*, neboli mohou se vyskytovat na levé straně pravidla, a díky tomu mohou být přepsány při derivování řetězce (i symboly + - lze použít jako nonterminální). Symboly označující práci se zásobníkem, představené v této kapitole, jsou ovšem *terminální* (nemohou se vyskytovat na levé straně pravidla) a při přepisování řetězce se vždy přepíší sami sebou (do P se přidají dvě pravidla typu: $[\rightarrow [$, $] \rightarrow]$, viz Definice 5).

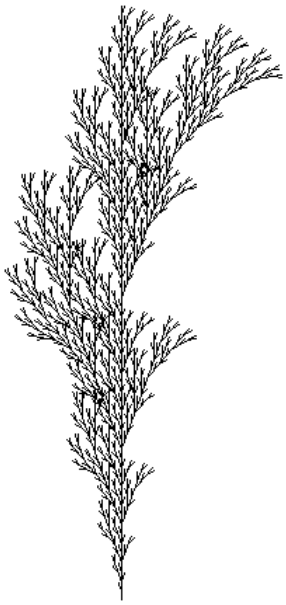
Na následujícím obrázku můžete vidět stromové útvary vygenerované využitím závorkových L-systémů. Všimněte si, že pro zdánlivě velkou složitost vygenerovaných struktur jsou pravidla zcela jednoduchá a vystačíme si s jedním nebo maximálně se dvěma přepisovacími pravidly.



n = 4, δ = 22.5°

ω: F

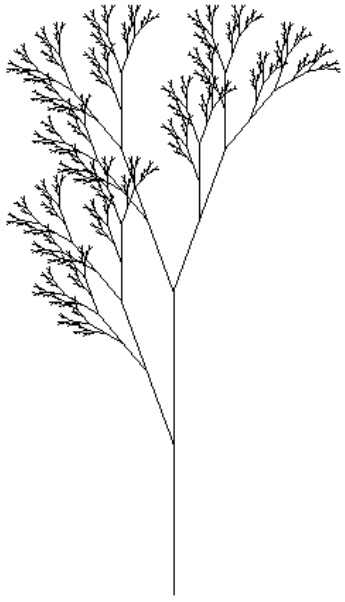
p: F → FF-[-F+F+F]+[+F-F-F]



n = 5, δ = 20°

ω: F

p: F → F[+F]F[-F][F]

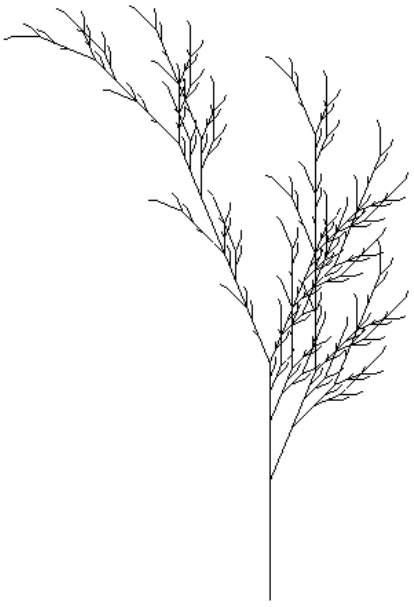


n = 7, δ = 20°

ω: X

p_1: X → F[+X]F[-X]+X

p_2: F → FF



n = 5, δ = 22.5°

ω: X

p_1: X → F-[[X]+X]+F[+FX]-X

p_2: F → FF

Obrázek 3.16 Příklady stromových struktur vygenerovaných závorkovými D0L-systémy.

3.6 Parametrické L-systémy

Velmi mocným rozšířením L-systémů pro modelování rostlinných organismů jsou parametrické L-systémy. Typy L-systémů představené v předchozích kapitolách jsou stále docela limitované, jelikož délka kroku d a úhel natočení želvy δ jsou pro všechny segmenty modelu konstantní. K řešení tohoto problému by se daly například přidat do abecedy symbolů V interpretovaných želvou nové symboly, které by znamenaly posunutí se vpřed o poloviční délku a podobně. Potom by však bylo psaní těchto systémů hodně složité a matematická krása, která je v nich obsažena, by se úplně vytratila. Takže abychom mohli právě tyto dřívější konstanty v průběhu vykreslování modelu želvou jednoduše měnit, byly zavedeny právě parametrické L-systémy, jejichž princip je docela jednoduchý na pochopení, avšak bohužel poněkud složitější na implementaci.

Jak již název napovídá, tyto L-systémy obsahují symboly, ke kterým jsou přidané určité parametry. Kombinaci symbolu a parametru nazýváme *modulem* [11]:

Definice 7 Parametrické L-systémy pracují s *parametrickými slovy*, jež jsou složené z řetězců modulů. Modul se skládá ze symbolu patřícího do *abecedy* V a parametrů z množiny *reálných čísel* $\mathbb{R}$. Modul, který obsahuje symbol $A \in V$ a parametry $a_1,a_2,\ldots,a_n \in \mathbb{R}$, se zapisuje stylem $A(a_1,a_2,\ldots,a_n)$. Každý modul patří do množiny $M = V \times \mathbb{R}^*$, kde $\mathbb{R}^*$ je množina konečných sekvencí parametrů. Množina všech řetězců modulů a množina všech neprázdných řetězců se značí jako $M^* = (V \times R^*)^*$ a $M^+ = (V \times R^*)^+$. Reálná hodnota *aktuálních* parametrů objevujících se ve slovech koresponduje s *formálními* parametry použitými ve specifikaci L-systému. Jestliže Σ je množina formálních parametrů, tak $C(\Sigma)$ označuje *logický výraz* s parametrem z Σ a $E(\Sigma)$ značí aritmetický výraz s parametry ze stejné množiny. Oba typy výrazů se skládají z formálních parametrů a číselných konstant, spolu kombinovaných pomocí aritmetických operací +,-,*,/,^ (3.1).

Dále se využívá omezení aplikace pravidla podle hodnoty parametru předchůdce při přepisování řetězce (podmínky). Zde se využívá relačních operací <,>,=,<=,>=, logických operací !, &, | (negace, logický součin, logický součet) a při vyhodnocování těchto operandů nabývá výsledek hodnoty *1* pro *pravdu* (*true*) a *0* pro *nepravdu* (*false*). Množiny všech správně zkonstruovaných logických a aritmetických výrazů s parametry z Σ se zapisují jako $\zeta(\Sigma)$ a $\xi(\Sigma)$.

Parametrický 0L-systém je podle [11] definován jako uspořádaná čtveřice $G = (V, \Sigma, \omega, P)$, kde:

- V je abeceda systému.
- Σ je skupina formálních parametrů.
- $\omega \in (V \times R^*)^+$ je neprázdná skupina parametrických slov nazývaná *axiom*.
- $P \subset (V \times \Sigma^*) \times \zeta(\Sigma) \times (V \times \xi(\Sigma))^*$ je konečná množina *pravidel*.

V zápisu pravidla se ještě využívá symbolů dvojtečky k oddělení levé strany pravidla od podmínky. Podmínka je nepovinná, a když ji není potřeba, tak se buď vůbec nepíše, nebo se do zápisu podmínky v pravidle píše hvězdička (*). Zápis pravidla poté vypadá následovně:

```
A(x,y) : x>7 → A(x+1,y+0.5)+B(x^0.5)CD∧E(1,x-y*2)          (3.1)
```

Pravidlo se aplikuje, pouze když je symbol modulu stejný jako symbol předchůdce v daném pravidlu, pokud je počet parametrů modulu stejný jako počet parametrů předchůdce v pravidlu a pokud je podmínka v pravidle vyhodnocena jako pravdivá (*true*). Hodnoty parametrů modulu jsou poté předány předchůdci pravidla, jímž se modul přepisuje a jsou uložené jako hodnoty proměnných tohoto předchůdce. Těmito hodnotami jsou poté nahrazeny proměnné v následníkovi pravidla.

Například pokud budeme chtít modul `A(9,2.5)` přepsat pravidlem (3.1), tak se prvně určí hodnoty proměnných v předchůdci. Neboli $x = 9$ a $y = 2.5$. Dále se zjistí, zda odpovídá podmínka, která pro $x > 7$ nabývá hodnoty *true* a pravidlo se může aplikovat. Výsledný řetězec bude následující: `A(10,3)+B(3)CD∧E(1,4).`

Definice 8 Jestliže modul a tvoří parametrický řetězec χ jako výsledek pravidla aplikovaného v L-systému G, zapisujeme jako $a \rightarrow \chi$. Nechť parametrický řetězec $\mu = a_1 \ldots a_m$ je libovolný řetězec nad V. Pak řetězec symbolů $\nu = \chi_1 \ldots \chi_m \in V^*$ je přímo derivován (generován) pomocí μ, tak zapisujeme $\mu \Rightarrow \nu$ právě tehdy, když $a_i \rightarrow \chi_i$ pro všechna $i = 1,\ldots,m$. Parametrický řetězec ν je generován z G derivací *délky n*, jestliže existuje vývojová sekvence řetězců $\mu_0,\mu_1,\ldots,\mu_n$ taková, že $\mu_0 = \omega$, $\mu_1 = \nu$ a $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$ [11].

Mějme parametrický L-systém:

```
ω:    A(3,2)B(5,1)
p1:   A(x,y) : x > 3   → B(x-1, y)
p2:   A(x,y) : x <= 3  → B(x+2, y*2)
p3:   B(x,y) : x < 3   → A(x+3, y-1)
p4:   B(x,y) : x >= 3  → A(x-y, 1)
```

poté pět derivačních kroků vypadá následovně:

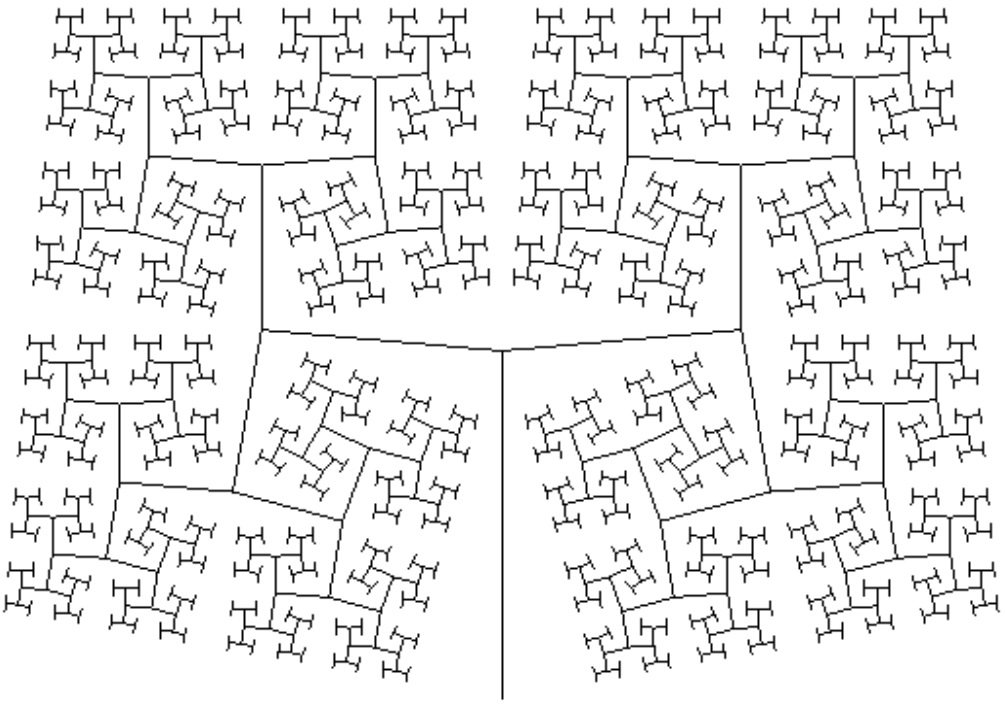
```
μ0:   A(3,2)B(5,1)
μ1:   B(5,4)A(4,1)
μ2:   A(1,1)B(3,1)
μ3:   B(3,2)A(2,1)
μ4:   A(1,1)B(4,2)
μ5:   B(3,2)A(2,1)
```

Pokud parametr je přidán k symbolu, jenž má pro interpretaci L-systému jistý grafický význam, tak jej hodnota prvního parametru tohoto modulu ovlivňuje následovně:

- `F(a)` : posunutí želvy vpřed o délku $a>0$ z původních souřadnic (x, y, α) na $(x+a*\cos\alpha, y+a*\sin\alpha, \alpha)$ a vykreslení úsečky mezi těmito body.
- `f(a)` : posunutí želvy vpřed o délku $a>0$ z původních souřadnic (x, y, α) na $(x+a*\cos\alpha, y+a*\sin\alpha, \alpha)$ bez kreslení úsečky mezi těmito body.
- `+(a)` : rotace želvy o úhel a s použitím rotační matice $R_U(a)$.
- `-(a)` : rotace želvy o úhel $-a$ s použitím rotační matice $R_U(-a)$.
- `&(a)` : rotace želvy o úhel a s použitím rotační matice $R_L(a)$.
- `∧(a)` : rotace želvy o úhel $-a$ s použitím rotační matice $R_L(-a)$.
- `\(a)` : rotace želvy o úhel a s použitím rotační matice $R_H(a)$.
- `/(a)` : rotace želvy o úhel $-a$ s použitím rotační matice $R_H(-a)$.

Dále se v L-systémech dá definovat konstantám určitý název. Překladač projde kód a nahradí všechny výskyty tohoto názvu za jeho hodnotu. Použití je následující: `#define <název> <hodnota>` a využívá se k definování konstant, které se vyskytují v kódu vícekrát a lze tak jednoduše a rychle změnit jejich hodnotu.

Pěkným příkladem parametrického L-systému je následující konstrukce převzatá z [11] (Obrázek 3.17). V tomto obrazci je rekurzivně zmenšovaná délka segmentu vykresleného želvou při interpretaci symbolu *F*.



```
#define R 1.456
n = 11, δ = 85°
ω:    A(1)
p:    A(s) → F(s)[+A(s/R)][-A(s/R)]
```

Obrázek 3.17 Větvící se struktura vygenerovaná parametrickým L-systémem.

Na obrázku je pěkně patrné, jak již jeho pravidla naznačují, že se s postupným derivováním zmenšuje délka segmentu, jenž se rozvětvuje. Neboli modul *F(s)* při derivačním kroku n_1 vykreslí úsečku o *R* delší než při kroku n_2.

Pokud bychom chtěli krokově simulovat růst daného obrazce, a ne aby jeho velikost byla již od první derivace stejná, tak můžeme upravit pravidla následovně, kde se hodnotou *R* nyní násobí a díky tomu se zvětšuje s každou derivací délka všech segmentů *F*:

```
#define R 1.456
n = 11, δ = 85°
ω:     A
p1:    A     → F(1)[+A][-A]
p2:    F(s) → F(s*R)
```

3.7 Stochastické L-systémy

Modely vygenerované jedním L-systémem jsou vždy totožné a kdybychom je chtěli spolu kombinovat na jedné scéně, docházelo by k umělé pravidelnosti. Proto se využívá *stochastických L-systémů*, které generují různé, ale přesto podobné modely.

Náhodnosti se dá dosáhnout dvěma způsoby. Za prvé pomocí náhodné interpretace symbolu s geometrickým významem (hodnoty délky kroku nebo velikost úhlu natočení apod.) a za druhé náhodností celého L-systému. První způsob samotný se spíše nepoužívá, protože díky němu nelze měnit topologii modelu. Druhým způsobem se dá změnit jak topologie, tak i geometrie. Jeho definice byla napsána pány Eichhorstem a Savitchem následovně [4]:

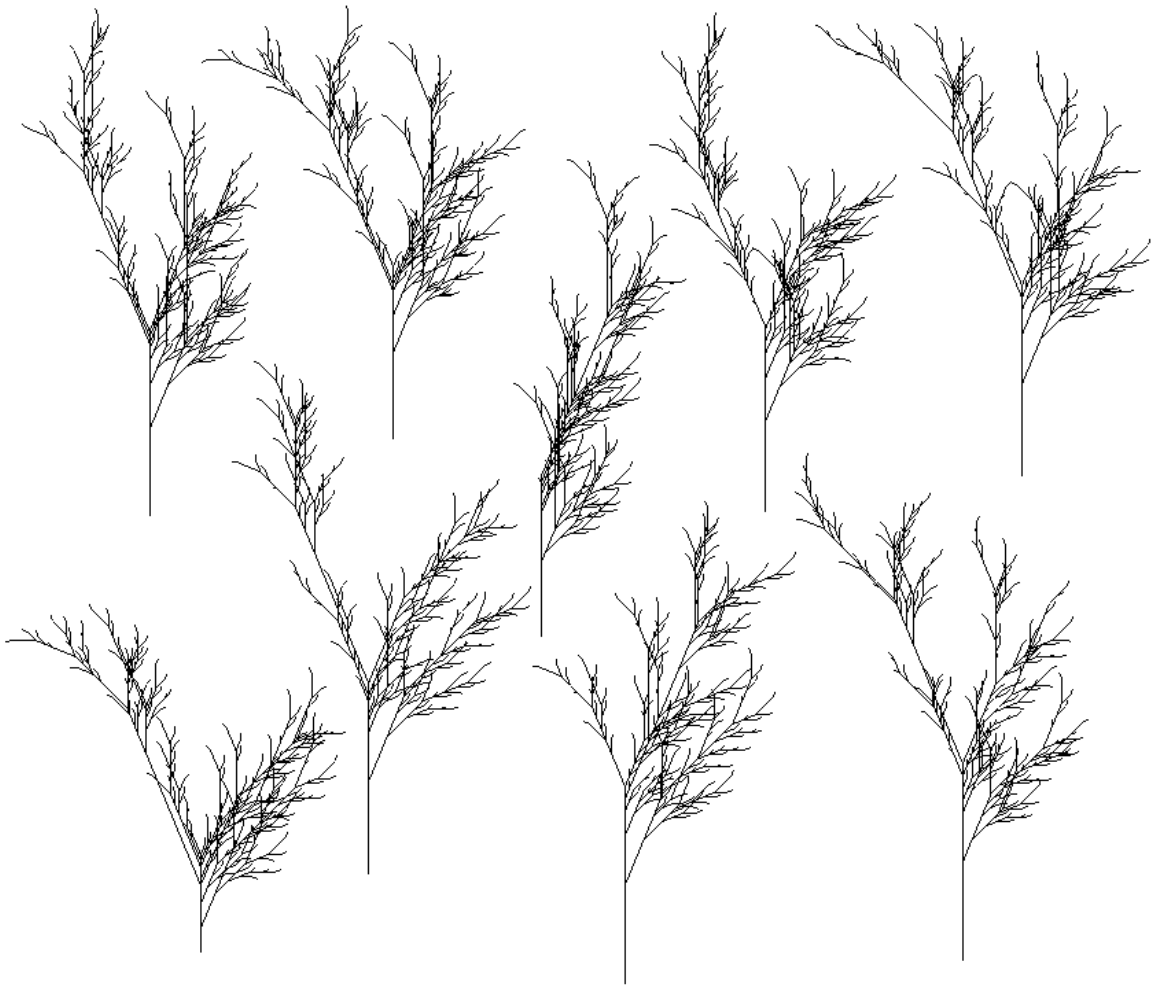
Definice 9 *Stochastický 0L-systém* je uspořádaná čtveřice $G = (V, \omega, P, \pi)$, kde *V* je abeceda symbolů, ω je axiom, *P* je množina pravidel, jejichž definice je stejná jako ta v kapitole 3.2.1. Funkce $\pi : P \rightarrow (0,1\rangle$, nazývaná jako *rozdělení pravděpodobnosti*, mapuje množinu pravidel na množinu *pravděpodobnostních pravidel*. To znamená, že pro každý symbol $a \in V$ je součet pravděpodobností všech pravidel s předchůdcem *a* roven jedné.

Definice 10 Derivací $\mu \Rightarrow \nu$ nazýváme *stochastickou derivací* v G_π pro každý výskyt symbolu *a* ve slově μ, když pravděpodobnost aplikace pravidla *p* s předchůdcem *a* je rovna $\pi(p)$. Takže rozdílná pravidla se stejným předchůdcem mohou být aplikované na různý výskyt stejného symbolu v jednom derivačním kroku.

Mějme zadán stochastický 0L-systém následujícího typu, který vychází z dříve představeného D0L-systému (Obrázek 3.16 vpravo dole):

$$
\begin{aligned}
p_1: &\quad X \xrightarrow{.80} F-[[X]+X]+F[+FX]-X \\
p_2: &\quad X \xrightarrow{.20} F-[[X]+X]+F[-FX]-X \\
p_3: &\quad F \xrightarrow{.90} FF \\
p_4: &\quad F \xrightarrow{.10} F
\end{aligned}
\qquad (3.2)
$$

kde $\cdot^{XX}$ označuje pravděpodobnost s jakou bude dané pravidlo aplikováno. Pravidla p_1 a p_2 představují větvení struktury, která se v 80% případů větví o kladný úhel a ve zbylých 20% o záporný úhel. Dále pravidla p_3 a p_4 znamenají růstový faktor, kdy v 90% segment rostliny zvětšuje svoji délku o dvojnásobek a ve zbylých procentech růst stagnuje. Modely vygenerované tímto stochastickým L-systémem vypadají například následovně:



Obrázek 3.18 Modely vygenerované stochastickým 0L-systémem (3.2).

4 Modelování listů rostlin

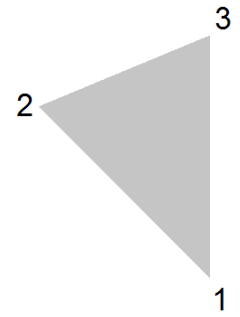
Jelikož listy rostlin zabírají plochu o určitém obsahu, tak nám k jejich modelování nestačí pouze úsečky vytvořené želví grafikou. Abychom mohli tento obsah graficky interpretovat, tak potřebujeme vykreslit *mnohoúhelníky* (*polygony*), jež budou vyplněné určitou barvou. Plocha listu se poté bude skládat z těchto na sebe navazujících mnohoúhelníků (nejčastěji trojúhelníků). Díky tomu je potřeba do L-systému přidat nové symboly, pomocí kterých tyto polygony v řetězci označíme a dané listy vykreslíme:

- `.` : označení vrcholu mnohoúhelníku na pozici, kde se želva momentálně nachází *(x, y, z)*.
- `{` : start mnohoúhelníku – následující vrcholy náleží k novému mnohoúhelníku.
- `}` : konec mnohoúhelníku – vykreslení plochy s vrcholy, které náleží novému mnohoúhelníku.

Dále se používá principu, že pokud je symbol *F* nalezen mezi složenými závorkami, tak se po jeho interpretaci přidá vrchol k mnohoúhelníku. A proto se do L-systému ještě musí přidat nový symbol *G*, který vykreslí úsečku stejně jako *F*, ale tento vrchol nepřidá (neboli zápis "*F*" je ekvivalentní k "*G*.").

Pokud bychom chtěli vykreslit jednoduchý trojúhelník, tak nám stačí například podobný zápis:

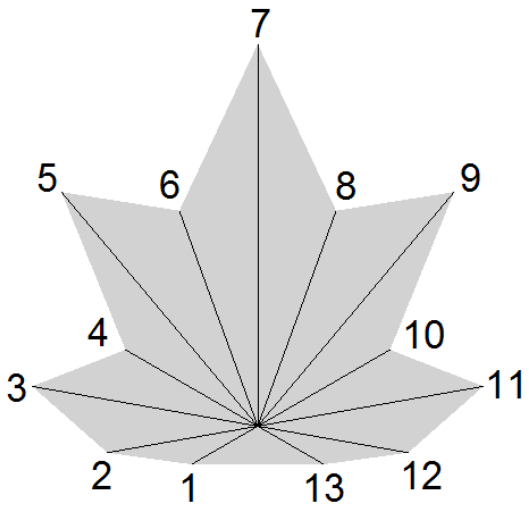
```
{.[+G.]G.}
 1   2  3
```



Vrcholy obrazce v řetězci jsou pro názornost označeny čísly. Želva začíná ve spodu obrazce a při interpretaci jako první najde symbol `{` a začne kreslit trojúhelník, neboli označí první vrchol polygonu (bod 1). Uloží svou pozici na zásobník, natočí se a posune do bodu 2, kde označí druhý vrchol. Načte svou pozici ze zásobníku a posune se vpřed do pozice označené bodem 3, kde je třetí poslední vrchol trojúhelníku. Symbol `}` značí vykreslení tohoto polygonu.

Na dalším obrázku je vymodelován list javoru, který je vygenerovaný pomocí následujícího řetězce a ve kterém jsou vrcholy postupně označeny čísly:

```
{[++++++G.][+++++GG.][++++GGG.][+++GG.][++GGGG.][+GGG.][GGGGG.]
          1          2          3         4          5         6         7
[-GGG.][--GGGG.][---GG.][----GGG.][-----GG.][------G.]}
      8         9        10         11         12         13
```



Obrázek 4.1 Příklad generování listu javoru.

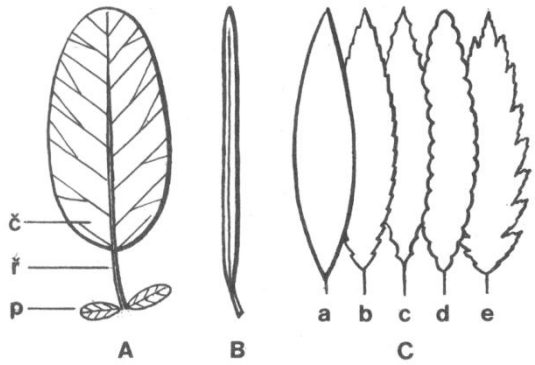
Jak je patrné, tak list je tvořen pouze z jednoho mnohoúhelníku (je celý ohraničen jedněmi složenými závorkami), kde pomocí zásobníku jsou označované jednotlivé vrcholy, jejichž pozice se vypočítává vždy z počáteční pozice L-systému (středu listu). Neboli po označení jednoho vrcholu se želva vždy vrátí na počáteční pozici. Tento princip je využit dále také u srdcovitého listu (Obrázek 4.5).

4.1 Dělení listů

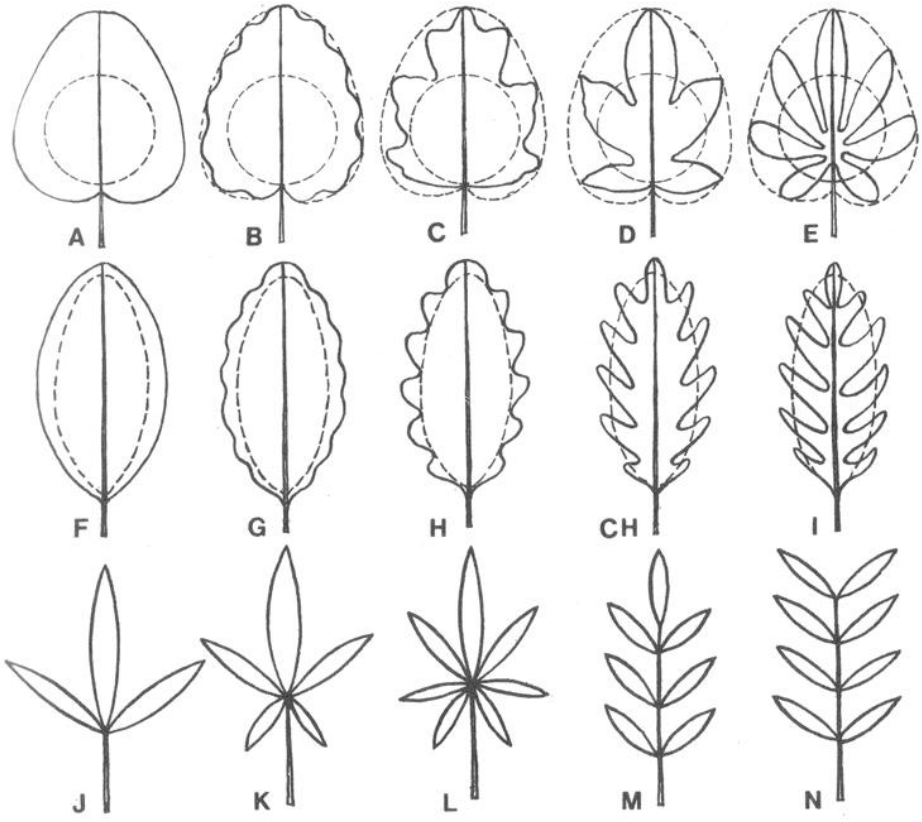
Listy se zpravidla skládají z čepele, řapíku a palistů (viz Obrázek 4.2A), ale existují i listy bez palistů a bez řapíku. Rozdělování listů se posuzuje podle více hledisek. Nejzákladněji se listy dělí podle členitosti na *jednoduché listy* a *složené listy* ([9] a [18]).

- Jednoduché listy jsou *nečlenité (celistvé)* nebo *rozčleněné zářezy v laloky*. Podle celkového obrysu a hloubky zářezů je dělíme na listy s *obrysem dlaně* (Obrázek 4.3A-E) a s *elipčitým obrysem* (Obrázek 4.3F-I).
- Složené listy mají čepel složenou z lístků vyrůstajících buď z jednoho místa, nebo po dvou proti sobě na hlavním vřetenu. Podle toho se rozlišují na listy *dlanitosložené* (Obrázek 4.3J-L) a listy *zpeřené* (Obrázek 4.3M-N).

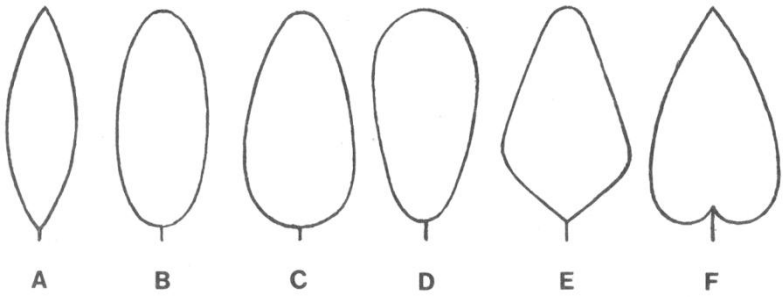
Podle velikosti plochy listu se dělí na *lupenité*, které mají čepel širší (Obrázek 4.2A) a *čárkovité* (Obrázek 4.2B), které jsou úzké. Podle okraje čepele rozlišujeme listy *celokrajné*, *pilovité*, *zubaté*, *vroubkovité*, *dvakrát pilové* (Obrázek 4.2 postupně Ca až Ce). Podle obrysu čepele rozlišujeme listy *kopinaté*, *elipčité*, *vejčité*, *obvejčité*, *kosníkovité* a *srdcovité* (Obrázek 4.4 postupně A až F).



Obrázek 4.2 Utváření listů převzaté z [9].



Obrázek 4.3 Dělení listů podle utváření čepele převzaté z [9].

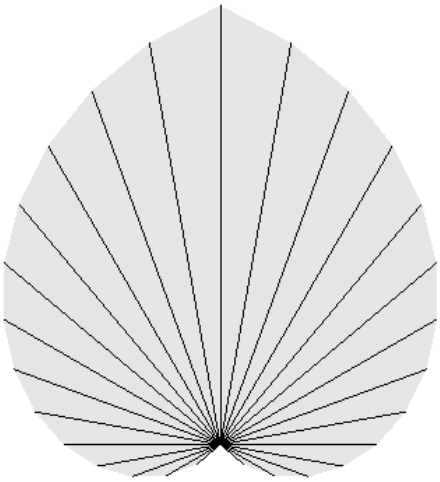


Obrázek 4.4 Dělení listů podle obrysu čepele převzaté z [9].

4.2 Modelování jednoduchých listů

Výhodou jednoduchých listů oproti složeným je, že generování řetězce pomocí pravidel, jež je tvoří, většinou netrvá příliš dlouho, a tudíž se snadno dají aplikovat ve větším množství na rozsáhlejší modely stromů nebo keřů a vygenerovat tak například strom hrušky pomocí srdcovitých listů.

Obrázek 4.5 ukazuje příklad takového srdcovitého listu. Můžeme na něm vidět, jak je jeho plocha vytvořena pomocí trojúhelníků vygenerovaných pravidly p_1 a p_2, kde každé z nich modeluje buďto levou, či pravou stranu listu (symbol A levou, protože rotuje o kladný úhel a symbol B pravou, protože rotuje o záporný). Dále potom pravidlem p_3 se při každé další derivaci postupně zvětšuje velikost trojúhelníků a tvoří tak tvar listu. U jednoduchých listů je možné simulovat jejich růst, když změníme počet derivačních kroků (viz Obrázek 4.6)



$n = 15,\ \delta = 10°$

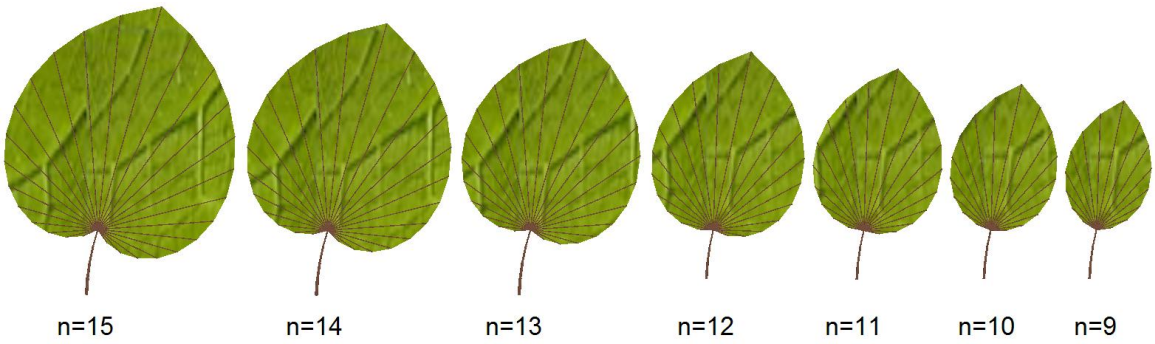
ω: [A][B]

p_1: A → [+A{.].C.}]

p_2: B → [-B{.].C.}]

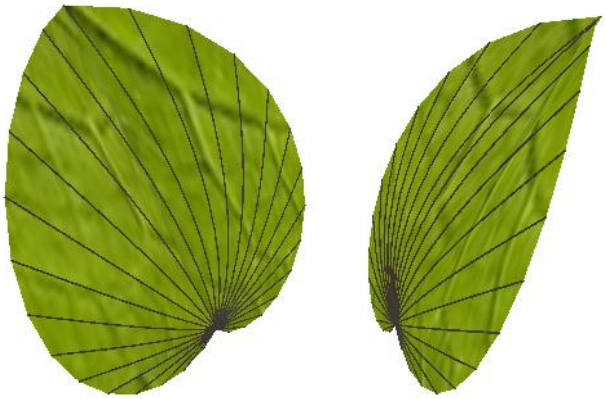
p_3: C → GC

Obrázek 4.5 Srdcovitý list vygenerovaný podle D0L-systému.



Obrázek 4.6 Vývoj otexturovaného srdcovitého listu.

Vygenerovaný list sice vypadá již docela pěkně a lze ho bez problému vyplnit libovolnou barvou, nebo pokrýt texturou, aby výsledek vypadal ještě reálněji. Pořád se ale zatím jedná o 2D model, jehož tloušťka je takřka nulová a při pohledu z boku bychom viděli pouze přímku. Abychom dosáhli více reálnějších výsledků, měli bychom při generování listů využít rotace želvy okolo dalších os a simulovat tak například natočení listu v 3D prostoru směrem k zemi, které vznikne vlivem gravitace. Úprava pravidel pro simulaci tohoto jevu je docela jednoduchá a může vypadat následovně:



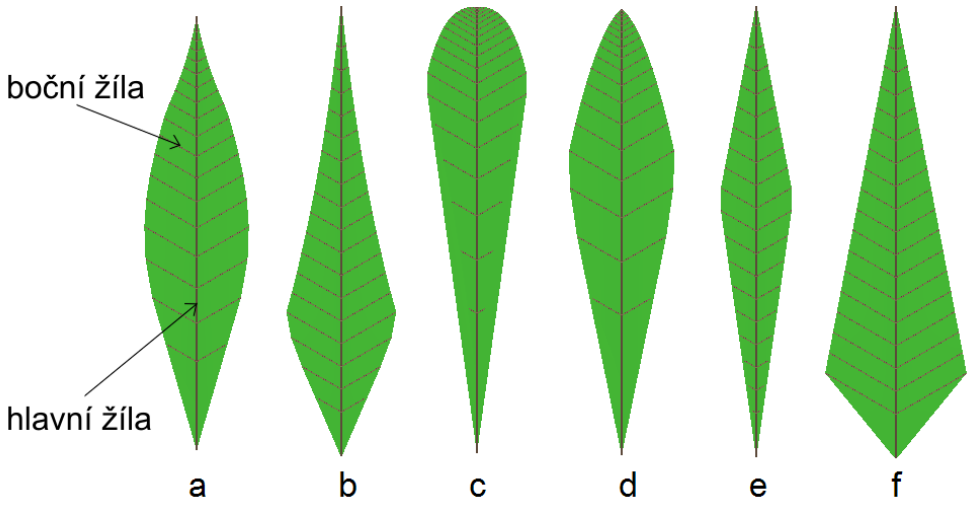
```
n = 15, δ = 10°
ω:      [A][B]
p1:     A → [+A{.].C.}]
p2:     B → [-B{.].C.}]
p3:     C → GC
p4:     G → G&(0.6)
```

Obrázek 4.7 Srdcovité listy vygenerované s využitím rotace v 3D prostoru.

Pro modelování podlouhlých jednoduchých listů, které mohou nabývat různých tvarů obrysů čepele (viz Obrázek 4.4), můžeme využít například následující D0L-systém, ve kterém se dají tyto tvary měnit pomocí modifikace hodnot parametrů. Jejich tvar úzce souvisí s růstovým faktorem jednotlivých segmentů a k jejich vygenerování jsou použita již parametrická pravidla. Tyto D0L-systémy mohou být využity například k modelování listů bezu, brusinky a všelijakých dalších rostlin a keřů.

Pravidlo p_1 (viz Kód L-systému 4.1) každým derivačním krokem rozšíří modul *A(t)*, jenž tvoří hlavní žílu listu (hlavní segmenty), o prodloužení tohoto segmentu *G(LA,RA)* a o další dva nové moduly *B(t)*, které tvoří boční žíly listu (boční segmenty) (Obrázek 4.8a). Ty jsou pomocí pravidla p_2 prodlouženy modulem *G(LB,RB)*. Jejich parametr *t* hraje důležitou roli růstového faktoru tohoto bočního segmentu listu. Je totiž v každém derivačním kroku postupně zmenšován o konstantu *PD* a díky tomu dříve nebo později nabude hodnoty menší než jedna a modul *B(t)* již nebude splňovat podmínku pro přepsání pravidlem p_2 (tím nebude dále růst jeho délka, nebudou se přidávat nové moduly *G(LB,RB)*). Každým dalším přepsáním modulu *A(t)* se zvýší hodnota jejího parametru (inicializačního parametru pro boční žíly) o jedničku, a tak se velikost celé boční žíly

(počet bočních segmentů) postupně ve směru od stonku zvětšuje. Tímto způsobem je simulováno počáteční rozšíření listu, které nabude své největší hodnoty v prostředku listu. Tento fakt je dán tím, že boční žíly vygenerované pozdějšími derivacemi sice mají parametr *t* větší než boční žíla ve středu, ale nejsou dosti staré na to, aby dokázaly dosáhnout její délky (neobsahují dostatek segmentů). Pomocí pravidla p_3 se ještě kontroluje růst segmentů *G(r,s)* (tloušťka, délka listu), kde je velikost všech segmentů s každým derivačním krokem vynásobena příslušnou růstovou konstantou (*RA* nebo *RB*). Tabulka 4.1 ukazuje hodnoty konstant pro změny tvaru těchto listů.



Obrázek 4.8 Jednoduché podlouhlé listy a jejich obrysy.

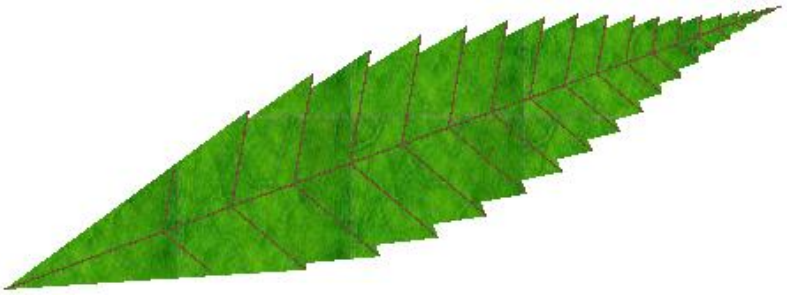
```
n = 20, δ = 60°

#define LA 5          /* úvodní délka hlavního segmentu */
#define RA 1.1        /* růst hlavního segmentu */
#define LB 1          /* úvodní délka bočního segmentu */
#define RB 1.2        /* růst bočního segmentu */
#define PD 1          /* zmenšování růstového faktoru */
ω:      {.A(0)}
p1:     A(t)          → G(LA,RA)[-B(t).][A(t+1)][+B(t).]
p2:     B(t)  : t>0   → G(LB,RB)B(t-PD)
p3:     G(s,r)        → G(s*r,r)
```

Kód L-systému 4.1 Jednoduchý podlouhlý list.

Tvar listu	LA	RA	LB	RB	PD
a	5	1.1	1.0	1.20	1.00
b	5	1.0	0.6	1.06	0.25
c	5	1.2	10.0	1.00	0.50
d	5	1.2	4.0	1.10	0.25
e	5	1.0	1.0	1.00	1.00
f	5	1.0	1.0	1.00	0.00

Tabulka 4.1 Hodnoty konstant pro generování jednoduchých podlouhlých listů.

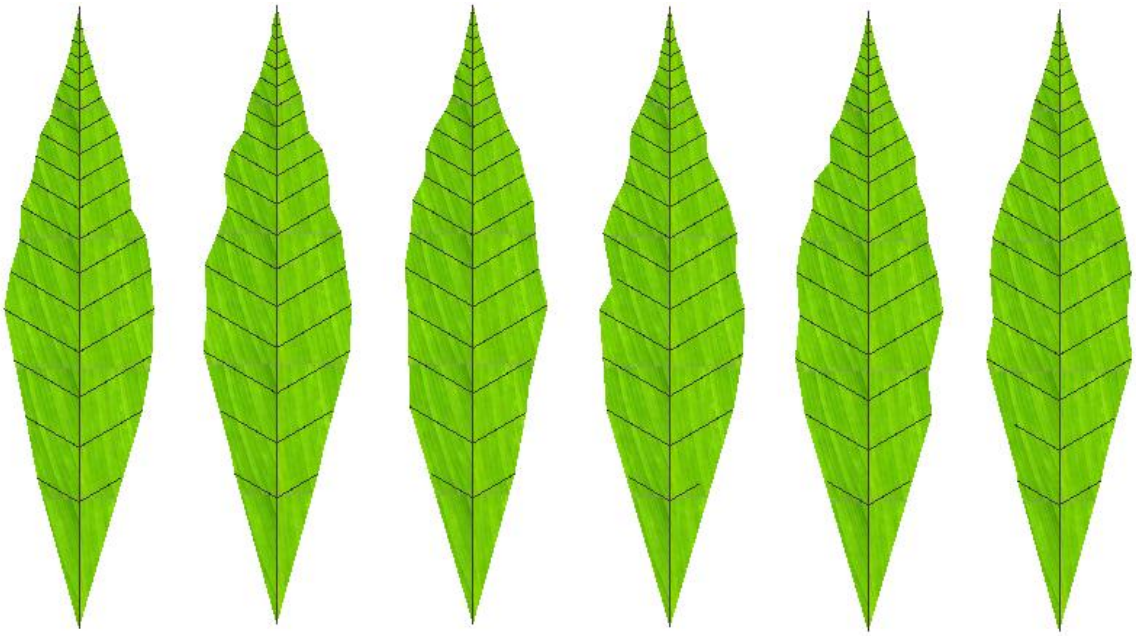
U podlouhlého listu je možnost změnit okraj z celokrajného tvaru na pilovitý. Tato změna je velmi jednoduchá a spočívá v záměně symbolu *G* v kódu za symbol *F* a tím se přidají další vrcholy do vykreslovaného polygonu (viz následující obrázek).



Obrázek 4.9 Jednoduchý podlouhlý list s pilovitým okrajem.

S využitím stochastických pravidel můžeme generovat zakřivení okraje tohoto listu tak, že pomocí náhodného přepsání modulu *B(t)* vybereme jednu z hodnot konstant *LBA* nebo *LBB*, kterou se prodlouží boční segment listu (viz Obrázek 4.10). Kód L-systému 4.1 je poté upraven například následovně:

```
#define LBA 1;
#define LBB 0.7;

p2:     B(t)  : t>0  :50→ G(LBA,RB)B(t-PD);
p3:     B(t)  : t>0  :50→ G(LBB,RB)B(t-PD);
```



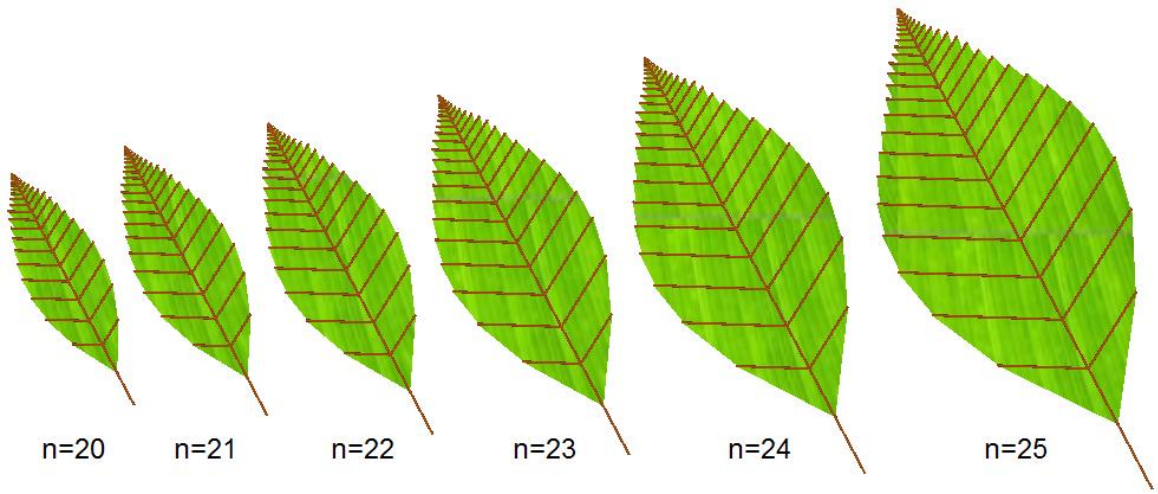
Obrázek 4.10 Podlouhlé listy se stochastickým tvarem okraje.

Poté můžeme ještě stochastický list převést do 3D prostoru a provést případné natočení jeho čepele úpravou pravidla p_3 obdobně jako u srdcovitého listu.

```
p3:     G(s,r)      → G(s*r,r)&(0.3);
```

Obrázek 4.11 Jednoduché podlouhlé listy v 3D prostoru se stochastickým okrajem.

Dalším pěkným příkladem listu s pilovitým okrajem je list růže (viz Kód L-systému 4.2 a Obrázek 4.12). V axiomu kódu jsou obsaženy dva moduly *A(0,0)* a *A(0,1)*, které rozlišují levou a pravou stranu listu (na každé straně želva rotuje opačným směrem). Toto rozlišení se provádí podmínkou u pravidel p_1 a p_2, kde se každé z nich aplikuje na daný modul, pouze tehdy, když je druhý parametr roven nule nebo jedné. Díky tomu je možné účelně tyto pravidla rozlišovat a rotovat želvou v potřebný okamžik určitým směrem. Modulem *G(LA,RA)* se zvětšují vzdálenost okraje listu od středu (list se rozšiřuje), modulem B(t) se tvoří samotné okraje listu a modulem *G(LC,RC,t)* výčnělky na okraji.

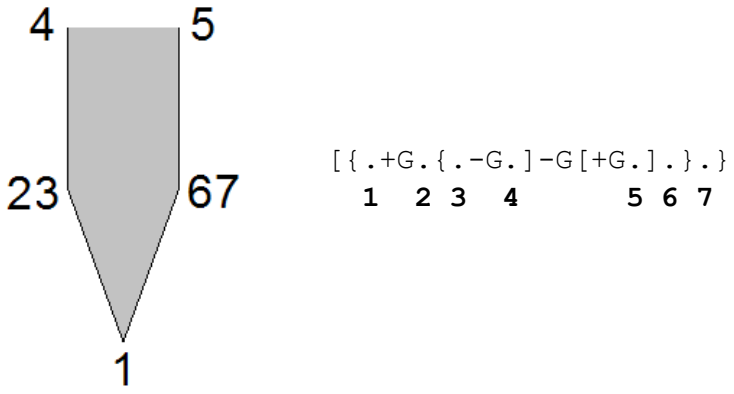


Obrázek 4.12 Vývoj listu růže.

```
n = 25, δ = 60°

#define LA 5          /* úvodní délka hlavního segmentu */
#define RA 1.15       /* růst hlavního segmentu */
#define LB 1.3        /* úvodní délka bočního segmentu */
#define RB 1.25       /* růst bočního segmentu */
#define LC 3          /* úvodní délka okrajového výčnělku */
#define RC 1.19       /* růst okrajového výčnělku */

ω:      [{A(0,0).}][{A(0,1).}]
p1:     A(t,d)      : d=0   → .G(LA,RA).[+B(t)G(LC,RC,t).}]
                              [+B(t){.]A(t+1,d)
p2:     A(t,d)      : d=1   → .G(LA,RA).[-B(t)G(LC,RC,t).}]
                              [-B(t){.]A(t+1,d)
p3:     B(t)        : t>0   → G(LB,RB)B(t-1)
p4:     G(s,r)      : *     → G(s*r,r)
p5:     G(s,r,t)    : t>1   → G(s*r,r,t-1)
```

Kód L-systému 4.2 List růže.

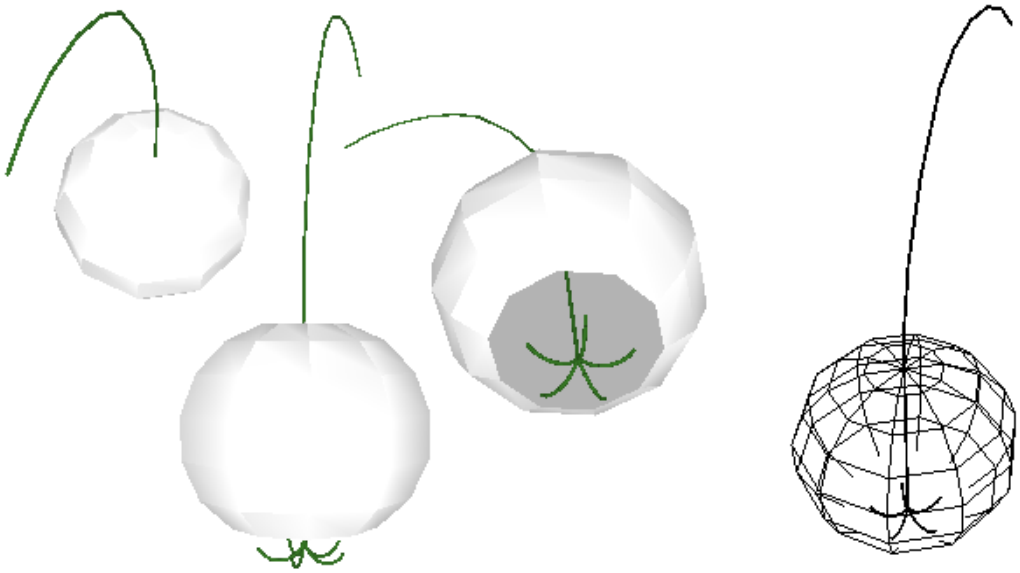
Předchozí příklady listů byly vytvořené stylem, kdy se v interpretovaném řetězci vyskytovaly plochy postupně za sebou, neboli vždy než se měla začít kreslit nová, tak předcházející byla již vykreslená. Pokud budeme chtít získat ještě o něco vypravovanější modely, tak můžeme do L-systému přidat zásobník vrcholů mnohoúhelníků. Díky němu budeme moci v řetězci polygony do sebe zanořovat, neboli budeme moct začít generovat nový, i když je stávající teprve rozpracován. Pokud tedy želva najde při průchodu řetězce symbol { a již generuje nějaký mnohoúhelník, tak se jeho vrcholy uloží na zásobník a začne se tvořit nový. Dále pokud najde } a na vrcholu zásobníku je uložen nějaký mnohoúhelník, tak vykreslí stávající, načte uložený a bude pokračovat v jeho generování. Při načtení symbolu tečky se však vrcholy přidávají pouze k mnohoúhelníku, který se aktuálně generuje.



Obrázek 4.13 Plocha vytvořená s využitím zásobníku mnohoúhelníků.

Obrázek 4.13 znázorňuje princip tohoto zanořování. Nejdříve se uloží počáteční pozice želvy na zásobník a začne se vykreslovat první mnohoúhelník. Označí se vrchol 1 a posune se na následující pozici, kde se označí vrchol 2. Nyní se želva nachází uprostřed vykreslování polygonu a nalezla v řetězci požadavek na vykreslení nového. Uloží tedy vrcholy 1 a 2 na zásobník a začne s novým vykreslováním. Označí si pozici vrcholu 3, posune se a označí vrchol 4. Nyní načte svou pozici ze zásobníku (nachází v bodu vrcholu 1). Posune se do bodu na obrázku označeném číslem 6, uloží svou pozici na zásobník a opět se posune. Označí vrchol 5, načte nedávno uloženou pozici a označí vrchol 6. Následuje vykreslení polygonu s vrcholy 3, 4, 5, 6. Nyní pokud želva zjistí, že zásobník mnohoúhelníků není prázdný, tak z něj načte uložené vrcholy 1, 2. Další její akcí je označení vrcholu 7 a vykreslení polygonu 1, 2, 7. Zásobník je nyní prázdný, takže se vykreslování ukončí.

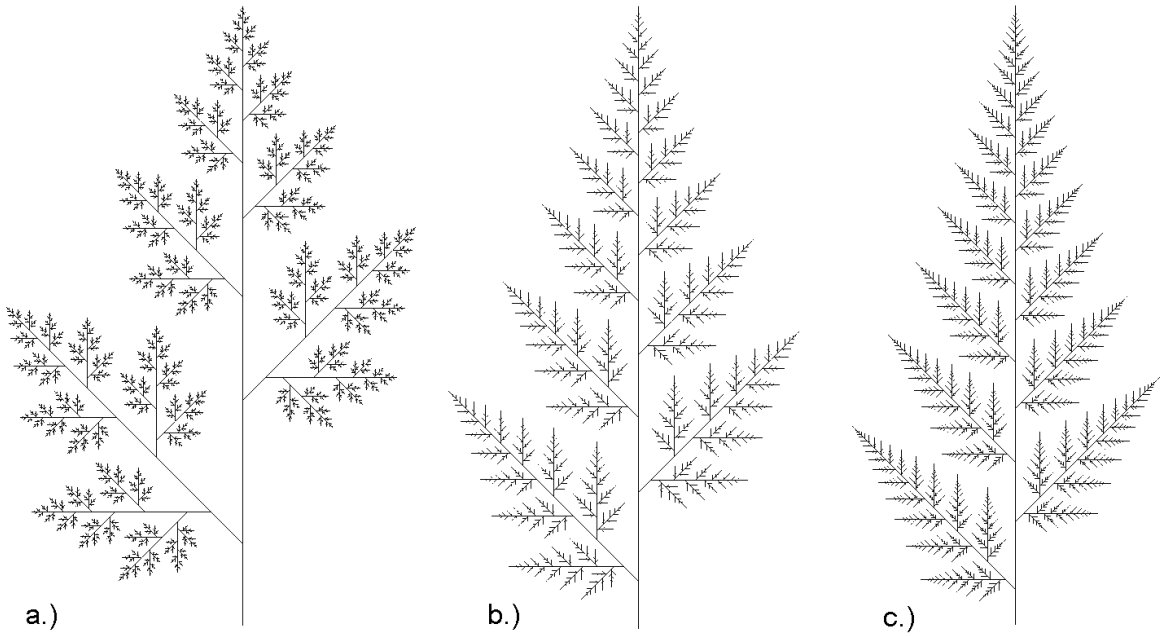
Obrázek 4.14 vyobrazuje květy konvalinky, pro jejichž vygenerování se využil zásobník vrcholů mnohoúhelníků. Vlevo jsou vyobrazeny plnohodnotné modely a vpravo drátěný model tohoto květu. Kód L-systému konvalinky je možno nalézt na přiloženém CD disku.



Obrázek 4.14 Květy konvalinky vytvořené pomocí zásobníku mnohoúhelníku.

4.3 Modelování složených listů

Dalším typem listů, pro jejichž modelování se využívá 0L-systémů, jsou *složené soběpodobné listy* (*compound leaves*). U těchto listů se již nevyužívá ploch (pevná souvislá plocha listu je totiž velmi malá), ale jejich problém spočívá v tom, že jsou vysoce soběpodobné a je u nich na první pohled vidět geometrický vztah mezi částí a celkem listu. Pomocí těchto L-systémů se generují listy okoličnatých rostlin jako mrkve, celeru, kmínu a podobně. Generování těchto listů je díky velkému počtu derivací velmi časově náročné, ale naštěstí se na rostlinách obvykle nevyskytuje ve větším množství. Podle postavení listu na stonku rozeznáváme složené listy *střídavé*, které stojí na stonku jednotlivě ve šroubovici a tedy v různých výškách, a listy *vstřícné*, které jsou vždy dva proti sobě.



Obrázek 4.15 Střídavé složené listy.

```
n = 20, δ = 45°

#define D 1              /* vrcholové zpoždění */
#define R 1.36           /* rádius růstu listu */

ω:     A(0)
p1:    A(d)   : d>0   → A(d-1)
p2:    A(d)   : d=0   → F(1)[+A(D)]F(1)B(0)
p3:    B(d)   : d>0   → B(d-1)
p4:    B(d)   : d=0   → F(1)[-B(D)]F(1)A(0)
p5:    F(a)   : *     → F(a*R)
```

Kód L-systému 4.3 Střídavý složený list.

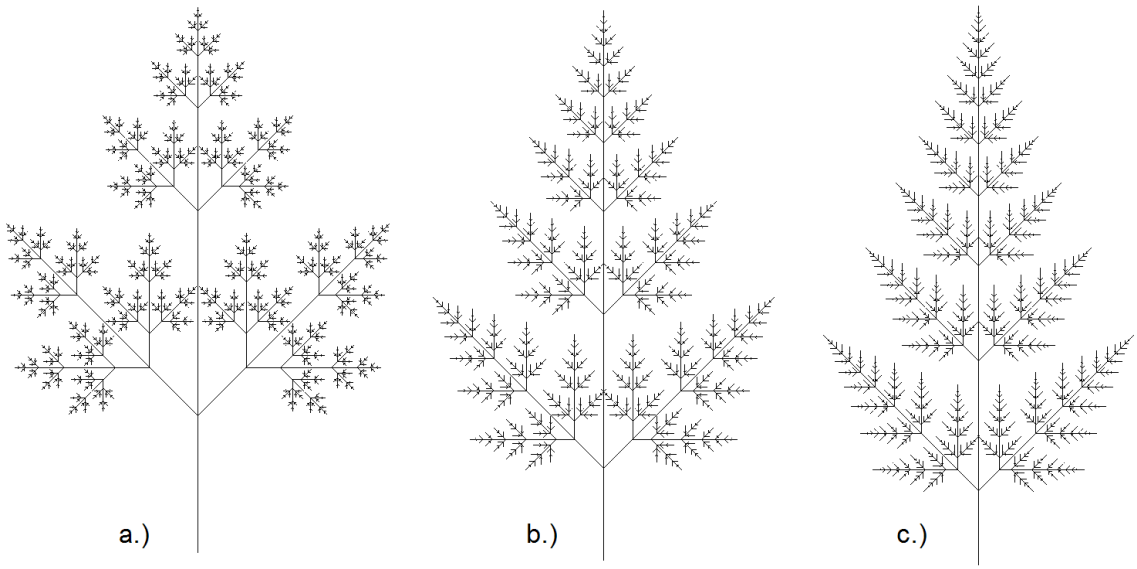
Obrázek 4.15 ukazuje příklad střídavého složeného listu vygenerovaného podle kódu 0L-systému (viz Kód L-systému 4.3). Tvar a hustota listu se mění změnou konstant a počtu derivací podle následující tabulky:

Tvar listu	**D**	**R**	**Počet derivací**
a	1	1.36	20
b	4	1.18	34
c	7	1.13	46

Tabulka 4.2 Hodnoty konstant pro generování střídavých složených listů.

Pomocí pravidla p_2 se přepíše modul *A(0)*, který vytvoří dva boční segmenty *A(D) B(0)* a prodlouží výchozí segment pomocí symbolů *F(1)*, neboli rozvětví otcovský segment na tři dceřiné. Pravidlo p_1 zpožďuje generování dalšího rozvětvení modulu *A(D)* o *D* kroků a pravidlo p_4 stejným způsobem jako p_2 rozvětví modul *B(0)*. Tímto způsobem se pravidelně střídají moduly *A(0)* a *B(0)* a díky tomu se vždy jeden z nich rozvětví a na druhý je aplikování zpoždění. Tento princip se provádí opakovaně, dokud není rozvětvení dostatečně husté. Pravidlo p_3 pomocí konstanty *R* prodlužuje segmenty a tím hlídá to, aby se navzájem neprotínaly.

Generování vstřícných složených listů je obdobné:



Obrázek 4.16 Vstřícné složené listy.

```
n = 10, δ = 45°

#define D 0          /* vrcholové zpoždění */
#define R 2.00       /* rádius růstu listu */

ω: A(0)
p1:    A(d)   : d>0   → A(d-1)
p2:    A(d)   : d=0   → F(1)[+A(D)][-A(D)]F(1)A(0)
p3:    F(a)   : *     → F(a*R)
```

Kód L-systému 4.4 Vstřícné složené listy

Tvar listu	D	R	Počet derivací
a	0	2.00	10
b	1	1.50	16
c	2	1.36	21

Tabulka 4.3 Hodnoty konstant pro generování vstřícných složených listů.

Obrázek 4.17 vyobrazuje stochastické modely složených listů, u kterých je míra zakřivení v důsledku gravitace, rozvětvení otcovského segmentu a prodloužení segmentů interpretováno náhodně.

Obrázek 4.17 Stochastické složené listy.

Pokud budeme chtít vygenerovat dlanitosložené trojčlenné, pětičlenné či sedmičlenné listy, tak můžeme využít podlouhlý list (Obrázek 4.8a) a kód jeho D0L-systému upravit tak, že vygeneruje více těchto listů najednou vedle sebe a přidá stonek (Obrázek 4.18). Aby byly listy na bocích menší než listy ve středu, tak se využívá tzv. *derivačního zpoždění*, které nezačne generovat list přímo při prvním derivačním kroku, ale například až při třetím (4.1). Kód tohoto L-systému je možno nalézt na přiloženém CD disku.

Obrázek 4.18 Trojčlenný, pětičlenný a sedmičlenný dlanitosložený list.

```
ω:    x;
p1:   x → y;                                    (4.1)
p2:   y → z;
p3:   z → list;
```

5 Modelování stromů

Prozatím jsme se zabývali jen úsečkami a plochami, které v podstatě nemají žádnou tloušťku, a nikde jsme nevěnovali tomu, jak vykreslit rostliny s nějakým tlustším kmenem. Jedním z nejjednodušších způsobů jak vygenerovat takový kmen je, že se zadá standardní šířka kmene, a potom místo interpretace symbolu *F* jako úsečky *ab* se vykreslí válec s touto šířkou z bodu *a* do bodu *b*.

Takto bychom ale vytvořili kmen, který má ve všech částech modelu stejnou šířku, což zřejmě nevystihuje biologickou podstatu přírody, kde větvička je většinou o hodně užší než kmen. Proto se tedy používá další symbol/modul, pomocí kterého se bude šířka segmentu postupně zmenšovat:

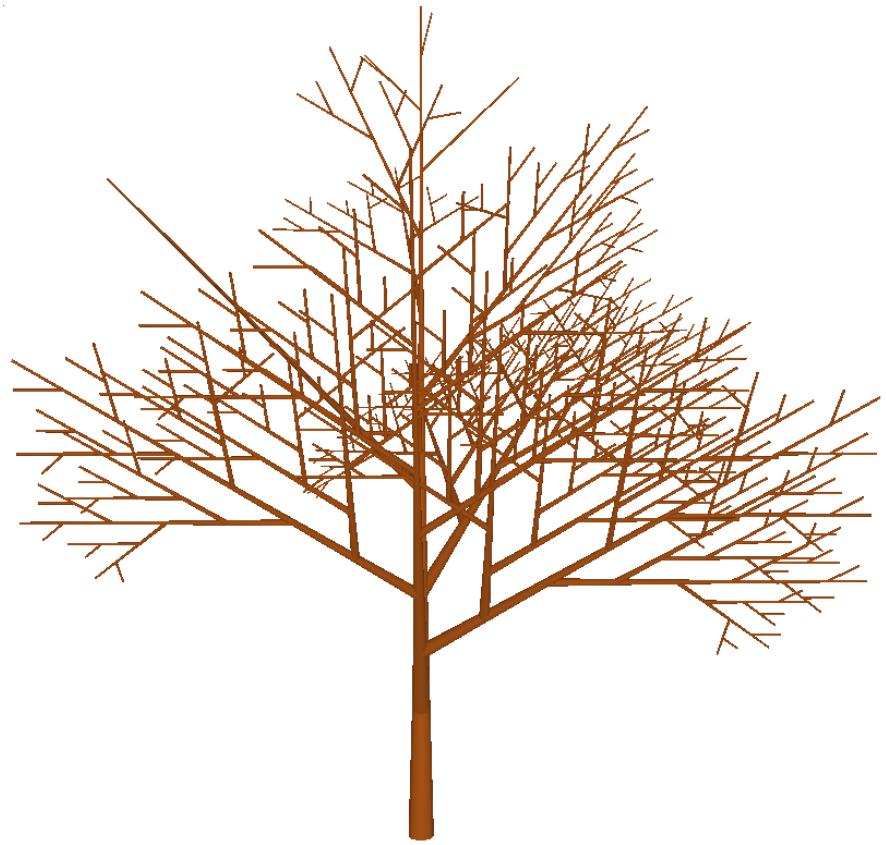
- `!` : zmenšení průměru následujících segmentů o definovanou standardní šířku.
- `!(a)` : zmenšení průměru následujících segmentů o hodnotu parametru *a*.

Jako první s modelováním stromů přišel pan Honda [5], který techniku modelování stromů definoval následovně:

- Segmenty stromu jsou rovné a jejich šířka se neuvažuje.
- Otcovský segment produkuje při jednom větvení dva dceřiné segmenty.
- Délka dvou dceřiných segmentů je zkrácena konstantami r_1 a r_2 podle délky otcovského segmentu.
- Otcovský segment a jeho dva dceřiné segmenty leží v jedné rovině a větvení se provádí podle úhlů a_1 a a_2 podle úhlu natočení otcovského segmentu.
- Větvící rovina musí fixně respektovat gravitaci, a tudíž být co nejblíže horizontální rovině.

Tato definice se v průběhu času ještě měnila a byly k ní přidány další principy ovlivňující topologii stromu (stochastický růst, tropismus, vír, atd.).

Kód L-systému 5.1 generuje stromový útvar vyobrazený na následující straně (Obrázek 5.1).



Obrázek 5.1 Stromový útvar s kmenem tvořeným válci.

```
n = 10

#define r1 0.9      /* stupeň zmenšování hlavního kmene */
#define r2 0.8      /* stupeň zmenšování segmentu větví */
#define a0 45       /* větvící úhel kmene */
#define a2 45       /* větvící úhel větví */
#define d  137.5    /* divergenční úhel */
#define wr 0.707    /* stupeň zužování kmene */

ω:     A(1,1.4)
p1:    A(l,w)  →   !(w)F(l)[&(_a0)B(l*r2,w*wr)]/(d)
                   A(l*r1,w*wr)
p2:    B(l,w)  →   !(w)F(l)[-(a2)C(l*r2,w*wr)]
                   C(l*r1,w*wr)
p3:    C(l,w)  →   !(w)F(l)[+(a2)B(l*r2,w*wr)]
                   B(l*r1,w*wr)
```

Kód L-systému 5.1 Stromový útvar s kmenem tvořeným válci.

6 Návrh a implementace interpretu stochastických 0L-systémů

V této kapitole je popsána implementace mé aplikace, která je vytvořená za účelem demonstrování využití stochastických bezkontextových L-systémů v počítačové grafice se zaměřením na generování listů rostlin. Aplikace zvládá všechny druhy L-systémů a jejich principy generování modelů, jež byly popsány v předchozích kapitolách. Všechny obrázky použité v této bakalářské práci jsou vygenerovány pomocí tohoto interpretu, pokud tomu není uvedeno jinak.

6.1 Syntaxe zdrojového kódu L-systému

Aby bylo možné pomocí aplikace vygenerovat modely popsané zdrojovým kódem, který obsahuje pravidla L-systému, tak se musí určit syntaxe a sémantika tohoto kódu:

- Na začátku kódu se definují hodnoty určitých klíčových slov, které určují například úhel rotace želvy, šířka segmentu apod. Jediným povinně definovaným klíčovým slovem je počet derivačních kroků, u ostatních se případně použije výchozí hodnota (viz Tabulka 6.1). Každá z definic musí být ukončená středníkem.
    - Syntaxe: `klicove_slovo = hodnota;`
- Dále lze na začátku kódu definovat názvy konstant (viz kapitola 3.6). Nezáleží na tom, zda se prvně uvedou hodnoty klíčových slov nebo definovaných konstant.
    - Syntaxe: `#define nazev hodnota;`
- Poté následuje klíčové slovo `axiom:` za nímž je řetězec symbolů ukončený středníkem značící počáteční řetězec daného L-systému.
- Za axiomem následuje klíčové slovo `rules:` za nímž jsou uvedeny pravidla oddělená středníky ve tvarech:
    - `pred -> succ;`
    - `pred : prob -> succ;`
    - `pred(par1,par2,…,parN) : cond -> succ;`
    - `pred(par1,par2,…,parN) : cond : prob -> succ;`

    kde `pred` značí předchůdce, `succ` následníka, `prob` pravděpodobnost aplikace pravidla, `parX` parametr předchůdce a `cond` podmínku pravidla. Pokud není pravděpodobnost zadaná, tak se pravidlu přiřadí výchozí hodnota 1 a pokud není zadána podmínka, tak je výchozí podmínky rovna * (*bez podmínky*). Šipka v pravidlech se píše pomocí spojení symbolů `->`, aby psaní kódu v programu nevyžadovalo žádné speciální znaky.

Název	Obor hodnot	Výchozí	Popis
`n`	$\mathbb{N} \geq 0$	-	Počet derivačních kroků
`delta`	$\mathbb{R}$	45°	Úhel rotace želvy kolem všech os
`lineDiv`	$\mathbb{R} \geq 0$	1	Podíl délky kroku želvy
`width`	$\mathbb{R} \geq 0$	1	Průměr kresleného segmentu
`widthDec`	$\mathbb{R} \geq 0$	none	Krok zmenšení průměru kreslené segmentu

Tabulka 6.1 Význam klíčových slov L-systému

Tabulka 6.1 ukazuje význam všech klíčových slov, které lze použít ve zdrojovém kódu L-systému. Hodnotou `lineDiv` se dělí délka kroku želvy, aby se výsledný model vešel do vykreslovací plochy, protože někdy je těžké odhadnout jeho velikost, která většinou závisí na počtu derivačních kroků a velikosti růstu segmentů (např. pravidlem $F(x) \rightarrow F(x*2)$ se s každým derivačním krokem velikost segmentu $F(x)$ zdvojnásobí). Hodnota `lineDiv` ovšem nijak neovlivňuje průměr kresleného segmentu `width`. Dále pokud klíčové slovo `width` není zadáno, tak se segmenty modelu kreslí pomocí úseček, pokud zadáno je, tak se kreslí pomocí válců s průměrem rovným hodnotě přiřazené tomuto klíčovému slovu.

6.2 Implementace aplikace

Interpret stochastických 0L-systémů je vytvořen v Microsoft Visual Studiu 2005 (Verze 8.0.50727.42) v programovacím jazyce C# s využitím .NET Frameworku (Verze 2.0.50727 SP2). Pro vykreslení vygenerovaných objektů je využita knihovna CsGL (C sharp Graphics Libraly verze 1.4.1), neboli balíček dobře známé knihovny OpenGL dovolující její využití v .NET Frameworku. Na přiloženém CD disku je možno nelézt instalátor aplikace, který sám nainstaluje všechny potřebné komponenty.

Aplikace se skládá z několika základních částí, které se dají rozdělit na uživatelské rozhraní, překladač (lexikální analyzátor, syntaktický a sémantický analyzátor), generátor řetězce (provedení derivačních kroků) a interpret výsledného řetězce symbolů.

6.2.1 Uživatelské rozhraní

Uživatelské rozhraní se skládá ze dvou oken, jež lze rozdělit na *editační* a *interpretační okno*. Hlavním prvkem editačního okna je *textový rámec*, ve kterém se spravuje samotný *kód L-systému* (viz syntaxe v kapitole 6.1 a příklad kódu v příloze C). Tento kód se dá libovolně načítat a ukládat do souboru pomocí příslušných tlačítek. Dále rozhraní obsahuje rámec, ve kterém se vypisují právě prováděné akce a jejich úspěšnost.

Kód se překládá pomocí tlačítka *Analyze*. Pokud se v kódu vyskytují nějaké chyby, tak se v textovém rámci vyznačí *červenou barvou* a pod ním se vypíše její *charakter*. Na tento výpis lze kdykoli kliknout myší a chyba v kódu se opět zvýrazní. Tlačítkem *Build* se generuje výsledný řetězec (pokud nebyl kód ještě přeložen, tak se přeloží). Generování probíhá na pozadí a je možno

ho kdykoli zastavit. Jeho průběh je uživateli znázorňován početně i graficky. Pomocí záložky nad textovým oknem se lze přepnout na zobrazení výsledného řetězce, který je pro názornost vypisován postupně ve všech derivačních krocích.

Dále se tlačítkem *Run* spouští samotný interpretovaný model, který se zobrazí v novém interpretačním okně (opět pokud nebyl kód ještě přeložen nebo nebyl vygenerován řetězec, tak se tyto akce provedou). Modelem lze libovolně posouvat, otáčet, oddalovat a přibližovat ho (popis ovládání je možno nalézt v příloze A). Aplikace dále nabízí uložení *bmp obrázku* modelu, zobrazení *FPS* (*Frames Per Second*) a zobrazení *drátěného modelu*. Plocha listů a kmenů modelu se vybarví vybranou barvou, nebo lze na tyto plochy nanést textury. Úprava barev a textur se provádí v editačním okně před spuštěním modelu. Textury, aplikované na modely v této bakalářské práci, jsou převzaté z [7].

Interpretačních oken lze v jeden moment spustit více, a pokud budeme chtít zobrazit různé modely vygenerované jedním stochastickým 0L-systémem, tak interpret nabízí možnost ponechat okno s již vyobrazeným modelem otevřené, opět tlačítkem *Build* vygenerovat řetězec a spustit nový model.

Aplikace má implementovanou práci s registry, a když uživatel zadá některou z cest k souboru (načtení kódu, načtení textury, uložení obrázku modelu atd.) nebo nastaví barvy objektů, tak se tyto informace do registrů uloží a při dalším spuštění aplikace se načtou. Díky tomu uživatel nemusí opět pracně hledat cesty k souborům nebo nastavovat barvy. Obrázky oken aplikace je možné nalézt v příloze A.

6.2.2 Lexikální analyzátor

Lexikální analyzátor (*scanner*) je nejjednodušší částí překladače a pracuje na principu *konečného automatu*, který přečte *tokeny* ze zdrojového kódu L-systému, určí jejich typy a hodnoty a předá je ke zpracování *syntaktickému analyzátoru* (viz následující kapitola). Token je souvislý blok textu, který má svůj typ a hodnotu. Například konstanta *5* je typu *integer* (*celé číslo*) s hodnotou *5*.

Lexikální analyzátor rozlišuje následující typy tokenů: *identifikátory* (proměnné), *konstanty* (celá čísla, desetinná čísla), *klíčová slova* (viz Tabulka 6.1), *matematické a logické operátory* a některé *speciální symboly* (`->`, `#define`, `axiom`, `rules`, atd.). Dále vynechává *bílé znaky* (mezery, nové řádky) a *komentáře*, které jsou v kódu ohraničení dvojicemi symbolů `/*` a `*/`. Tyto sekce kódu slouží pouze pro přehlednost kódu a pro samotný překlad nemají význam. Lexikální analyzátor dále rozeznává tokeny, které nejsou v kódu L-systému přípustné.

6.2.3 Syntaktický analyzátor a sémantický analyzátor

Syntaktický analyzátor (*parser*) je zase nejtěžší částí celého programu. Jeho účelem je analyzování vstupního kódu, který získává ve formě *tokenů* od lexikálního analyzátoru, a jeho transformování do datové struktury (nejčastěji derivačního stromu), která je vhodná pro pozdější zpracování pomocí generátoru. Syntaktický analyzátor pracuje na principu shora dolů, neboli analýza začíná od kořene stromu a postupuje zleva doprava dolů. Syntaktickou analýzu zdola nahoru, která se používá pro vyhodnocování výrazů, ve svém programu nepoužívám, protože matematické

operace, jež se využívají pro změny parametrů modulů L-systému, jsou omezené pouze na aritmetické operace (+,-,*,/), které se dají vypočítat tak, že se prvně vynásobí nebo vydělí potřebná čísla a poté se provede sčítání a odčítání výsledků těchto operací se zbylými čísly.

Ve své aplikaci ukládám data do několika polí tříd, kde každé pole je specifické pro data v něm uložená (pole klíčových slov, pole definovaných konstant, pole pravidel a samotnou třídu pro axiom). Pole definovaných konstant se používá pouze pro nahrazení názvů těchto konstant jejich hodnotou v pravidlech na konci parsování. Další pole jsou využita pro generování a grafickou interpretaci řetězce.

Dalším úkolem syntaktického analyzátoru je kontrola, zda součet pravděpodobností stochastických pravidel je roven jedné, a pokud se překlad nezdaří tak určit, kde nastala chyba a informovat o tom uživatele pomocí zvýraznění tohoto úseku v kódu L-systému. Účelem sémantického analyzátoru je kontrola oboru hodnot jednotlivých klíčových slov (viz Tabulka 6.1)

6.2.4 Generátor řetězce

Pomocí generátoru se generuje výsledný řetězec, který se použije k vymodelování obrazce pomocí interpretu. Generování začíná paralelním přepsáním symbolů v řetězci axiomu pomocí zadaných pravidel (viz kapitola 3.2). Přepsání se provádí opakovaně po počet uživatelem zadaných derivačních kroků, který je syntaktických analyzátorem uložen v poli klíčových slov.

Jeden derivační krok probíhá ve dvou průchodech celého řetězce. V prvním průchodu se určí symbol/modul v řetězci, který se může vyskytovat na levé straně pravidla. Pokud se jedná o modul, tak se načtou jeho parametry a uloží se seřazené podle jejich pozice do pole parametrů. Dále se určí pravidla, která danému symbolu/modulu vyhovují (je stejný jako předchůdce pravidla) a ujistí se, u kterých je platná podmínka. Pokud je takových pravidel více než jedno, tak se sečtou jejich pravděpodobnostní koeficienty a pomocí funkce *Random()* se vygeneruje náhodné číslo v rozsahu *0..součet*, a tím se určí právě jedno z těchto pravidel. Symbol/modul se poté přepíše pravou stranou vybraného pravidla. Pokud je vyhovující pravidlo pouze jedno, tak se symbol automaticky přepíše jím, a pokud nevyhovuje žádné, tak symbol v řetězci zůstává nepřepsán (viz Definice 5). V druhém průchodu řetězce se najdou všechny kulaté závorky, a pokud jsou v nich obsažené některé aritmetické operace, tak se vypočítá výsledek daného výrazu.

Největším problémem generování je, že délka výsledného řetězce může s každým derivačním krokem růst až exponenciálně v závislosti na délce pravých stran pravidel. Díky tomu roste i čas tohoto generování, který může v některých případech dosáhnout až desítek minut. Například u generování modelu složeného listu z kapitoly 4.3 (Obrázek 4.15). Tato vlastnost je ale pouze daní za to, že pravidla L-systému jsou tak jednoduchá.

6.2.5 Interpret výsledného řetězce symbolů

Při následné interpretaci modelů se prochází celý výsledný řetězec, a když se najde symbol, jenž má grafický význam, tak se interpretuje pomocí akce s ním spojené. Pokud bychom ovšem procházeli celý řetězec při každém překreslení obrazovky, tak by při složitějších modelech docházelo k sekání obrazu v důsledku vytížení procesoru počítače. Kvůli tomu se průchod

řetězcem provede při interpretaci pouze poprvé a to tak, že se vygenerovaná grafická primitiva uloží do tzv. *pole vrcholů* (*vertex arrays*), ze kterého je následně možné pomocí několika málo volání funkcí tato primitiva vykreslit. Výhodou tohoto přístupu je, že procesor nemusí provádět složité výpočty (rotace, posuny, apod.) a v ideálním případě jsou data uložena přímo v paměti grafického akcelerátoru a není tudíž nutné tyto data při každém překreslení přenášet po sběrnici, což je v dnešní době také jedna z nejvíce zpomalujících operací při vykreslování třírozměrných scén [19]. Fakt, že pomocí pole vrcholů se celá aplikace zrychlí, potvrzuje i to, že před jeho implementací v interpretu se u složitějších modelů pohyboval počet snímků za sekundu okolo 10-20 a poté u stejných modelů tato hodnota vzrostla na 140-160 FPS.

Bohužel se mi žádným způsobem nepodařilo do pole vrcholů uložit pozice válců, proto když uživatel zadá vykreslení kmene modelu válci, tak se při každém překreslení scény tyto pozice počítají. Překreslování se provádí pouze v nutných případech (při uživatelem inicializovaných rotacích výsledné scény a podobně), což vede také k menším nárokům na výkon.

7 Možné pokračování projektu

7.1 Změna barev v průběhu generování

Rozlišení barev modelu na barvu plochy a barvu kužele v některých případech nemusí být dostačující, neboť například můžeme chtít vygenerovat květ rostliny pomocí ploch a nechceme, aby měl stejnou barvu jako listy. Aby bylo možné v průběhu generování modelu měnit barvu všech segmentů, tak můžeme do kódu L-systému přidat nový modul, který bude tuto barvu interpretovat. Například zápisem `c(red, green, blue)`, kde parametry by značily intenzitu daných složek barvy.

Princip by byl takový, že po nalezení modulu by se změnila barva všech následujících segmentů, dokud by se nenašel nový modul s jinou barvou. Hodnota aktuální barvy by se musela ukládat také na zásobník, aby se po vykreslení objektu jinou barvou nemuselo vše nastavovat zpět.

7.2 Triangulace rotačního komolého kužele

Nyní mnou naprogramovaný interpret stochastických 0L-systémů pro generování kmene rostliny využívá válců, které jsou vygenerované pomocí jedné funkce grafické knihovny OpenGL. Bohužel pro tyto válce neexistuje způsob, jak je uložit do pole vrcholů popsaného v kapitole 6.2.5, a tedy se musí jejich pozice při každém překreslení scény znova počítat. Aby bylo možné tedy celou aplikaci ještě více urychlit, tak by kmen rostliny mohl být interpretován pomocí rotačního komolého kužele, pro jehož výpočet potřebujeme znát pouze průměry podstav a středy podstav.

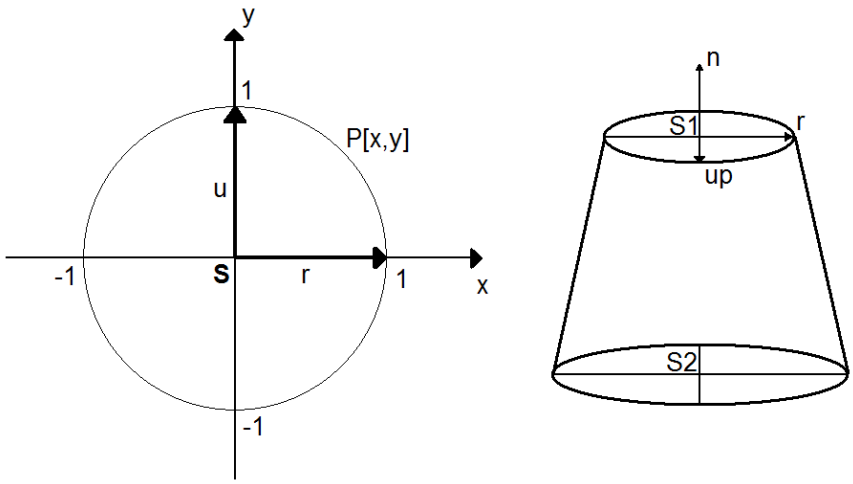
Princip je takový, že se vypočítají body na kružnicích podstav kužele, které se následně označí jako vrcholy polygonu. Při generování pláště se poté jednoduše spojí protilehlé body obou kružnic a při generování podstav společné body jedné kružnice. Body na kružnici v prostoru se vypočítají následovně:

$$P = S + \vec{R} * r_k$$

kde P je vypočítaný bod, S je střed kružnice, r_k je poloměr kružnice a $\vec{R}$ je jednotkový vektor:

$$\vec{R} = \frac{\vec{r} * x + \vec{u} * y}{\left|\vec{r} * x + \vec{u} * y\right|}$$

Ze středů obou kružnic (podstavy a vrcholu) se vypočítá normálový vektor kolmý na kružnici $\vec{n} = S_1 - S_2$. Potom $\vec{r}$ je jakýkoliv vektor kolmý na $\vec{n}$ a $\vec{u} = \vec{r} \times \vec{n}$. Dosazením vhodných x a $y \in \langle -1;1 \rangle$ do vzorce se dostanou jednotlivé body na kružnici (Obrázek 7.1).



Obrázek 7.1 Triangulace rotačního komolého kužele.

7.3 Kontextové L-systémy

Kontextové L-systémy (*context-sensitive L-systems*) se liší od bezkontextových tím, že berou v úvahu symboly okolo předchůdce pravidla při přepisování řetězce. Tento princip nachází využití pro simulování interakce mezi částmi rostliny. 2L-systémy využívají zápis předchůdce pravidla stylu `lc < pred > rc` → `succ`, kde se `pred` přepíše pouze tehdy, když je v řetězci předcházen symbolem `lc` a následován symbolem `rc`. Takže symboly `lc` a `rc` značí levý a pravý kontext pravidla. 1L-systém má omezenou pouze jednu stranu předchůdce a nezáleží na tom kterou. Pravidla se značí stylem `lc < pred` → `succ` nebo `pred > rc` → `succ`. Pomocí 2L-systémů se dají například vygenerovat následující stromové struktury.

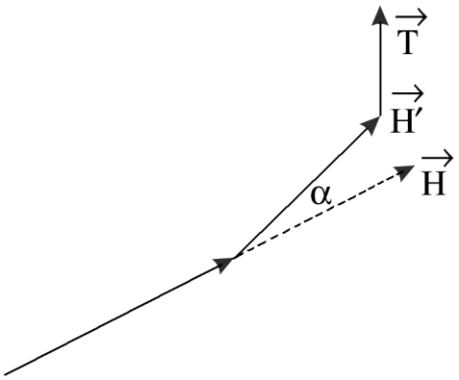
Obrázek 7.2 Stromové struktury vygenerované pomocí 2L-systémů převzaté z [11].

7.4 Předdefinované plochy

Pokud bychom chtěli modelovat scény, na kterých se budou vyskytovat některé objekty vícekrát (například stále stejný list na stromech), tak je zbytečné, aby se pro každý takový objekt pomocí derivací vytvářel nový řetězec symbolů. Proto je lepší mít tuto plochu uloženou například někde v souboru. Soubor se poté pomocí předdefinovaného klíčového slova může načíst například následovně: `#include <cesta> <nazev>`. Kde `#include` značí import, `<cesta>` označuje cestu k souboru a `<nazev>` název souboru, pod kterým se bude v řetězci tato plocha vyskytovat. Poté se do L-systému musí přidat ještě modul `~(<nazev>)`, který když želva v řetězci najde, tak vykreslí předdefinovanou plochu daného názvu. V souboru musí být kromě řetězce generujícího plochu, také uveden i kontaktní bod plochy a vektory určující orientaci plochy. Plocha je poté posunuta do požadovaného bodu a je natočena podle rotace želvy. U této plochy se dá měnit její barva i velikost, ale její tvar vždy zůstává stejný [19].

7.5 Tropismus

Tropismus je technika, která zásadním způsobem ovlivňuje topologii stromů. Zohledňuje sílu, která se dá popsat například jako gravitace či vliv slunce. Při aplikaci tohoto principu se po vykreslení každého segmentu stromu želva natočí o malý úhel ve směru předdefinovaného vektoru $\vec{T}$. Tento úhel se vypočítává z rovnice $\alpha = e|\vec{H} \times \vec{T}|$, kde e je hodnota určující, jak moc se bude želva natáčet.



Obrázek 7.3 Korekce segmentu s orientací $\vec{H}$ díky vektoru $\vec{T}$.

8 Závěr

Tato bakalářská práce se zabývala převážně stochastickým modelování listů s využitím 0L-systémů a byly v ní popsány principy modelování téměř všech existujících tvarů listů. Dále zde byly uvedené způsoby generování a interpretace modelů pomocí hlavních druhů 0L-systémů v rozsahu od nejjednodušších lineárních až po složitější stromové struktury. Práce obsahuje také letmý úvod do fraktální geometrie, jejíž hlavním aspektem je soběpodobnost, která se hojně využívá u modelování složených listů. Dále je v práci nastíněno modelování květů pomocí zásobníku mnohoúhelníků a stromů s určitou tloušťkou kmene.

Aplikace stochastických pravidel na modely listů byla vesměs úspěšná a vznikly tak druhy listů, které se dají bez problémů ve větším množství aplikovat na modely složitějších stromových struktur a dosáhnout tak reálnějších výsledků, které jsou zapříčiněné tím, že listy jsou vůči sobě podobné, ale žádný z nich není zcela stejný. Stochastická pravidla byla aplikována i na samotné stromové útvary, takže u nich nebyly změněny pouze tvary listů, ale i celková topologie a geometrie jejich kmene (viz Příloha D). Tudíž s pomocí pravidel jednoho stochastického L-systému v kombinaci s moderní počítačovou grafikou lze vygenerovat rozsáhlou scénu s rostlinami, která se nebude v žádném případě uměle opakovat, což je velkým přínosem této bakalářské práce.

Samotná aplikace nabízí uživateli velkou škálu možností zobrazení výsledného modelu a to v různých úhlech pohledu a ve zvolených barvách či texturách. Interpret dále umožňuje vizuální porovnání výsledků v reálném čase, kdy zobrazí více vygenerovaných modelů současně. Ke konci práce byl nastíněn další možný vývoj projektu zaměřený na modernější způsoby generování L-systémů a rozšíření možností změny vzhledu modelu v průběhu generování řetězce pomocí pravidel.

Literatura

[1] Abelson, H., diSessa, A. A. *Turtle geometry*. M.I.T. Press, Cambridge, 1982.

[2] Backus, J. W. *Programování v jazyku ALGOL 60*, Praha, 1963.

[3] Campos, A. M. *Drawing explaining Fractal fern IFS (png version)*. Wikipedie, otevřená encyklopedie. Poslední modifikace: 16. května 2006 [cit. 2009-04-08]. Dostupné na URL <http://cs.wikipedia.org/wiki/Soubor:Fractal_fern_explained.png>

[4] Eichhorst, P., Savitch, W. J. *Growth functions of stochastic Lindenmayer systems*. Information and Control, 45:217–228, 1980.

[5] H. Honda. *Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body*. Journal of Theoretical Biology, 31:331–338, 1971.

[6] Chomsky, N. *Three models for the description of language*. IRE Transations on Information Theory, ročník 2, č. 3, s. 113–124, 1956.

[7] Image * After, *The raw base for your creativity*, [cit. 2009-05-09]. Dostupné na URL <http://www.imageafter.com/>

[8] Mandelbrot, B. B. *The fractal geometry of nature*. Freeman, W. H., San Francisco, 1982.

[9] Martinobský, J., Pozděna, M. *Klíč k určování stromů a keřů.* Státní pedagogické nakladatelství. Praha. 1987

[10] McKenna, D. M. SquaRecurves, E-tours, eddies and frenzies: *Basic families of Peano curves on the square grid. In Proceedings of the Eugene Strens Memorial Conference on Recreational Mathematics and its History*, 1989. To appear.

[11] Prusinkiewicz, P., Lindenmayer, A. *The Algorithmic Beauty of Plants*. New York: Springer, 1990, ISBN 0-387-97297-8.

[12] Rozenberg, G., Salomaa, A. *The mathematical theory of Lsystems*. Academic Press, New York, 1980.

[13] Salomaa, A. K. *Formal Languages.* Academic Press, 1973.

[14] Stevens, R. J., Lehar, A. F., Perston, F. H. *Manipulation and presentation of multidimensional image data using the Peano scan*. IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-5(5):520–526, 1983.

[15] Techet, J., Masopust, T., Meduna, A. *Modern Formal Language Theory*. Poslední modifikace 2009 [cit. 2009-05-04]. Dostupné na URL <http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?id=lectures%3Aphd%3Atid%3Atid&cache=cache&media=lectures:phd:tid:frvs:08-lsystemspres.pdf>

[16] Tišnovský, P. *Seriál Fraktály v počítačové grafice*. Poslední modifikace 12.6.2007 [cit. 2009-04-03]. Dostupné na URL <http://www.root.cz/serialy/fraktaly-v-pocitacove-grafice/>

[17] Tišnovský P. *Seriál Letní škola programovacího jazyka Logo*. Poslední modifikace 6.11.2007 [cit. 2009-04-28]. Dostupné na URL <http://www.root.cz/serialy/letni-skola-programovaciho-jazyka-logo/>

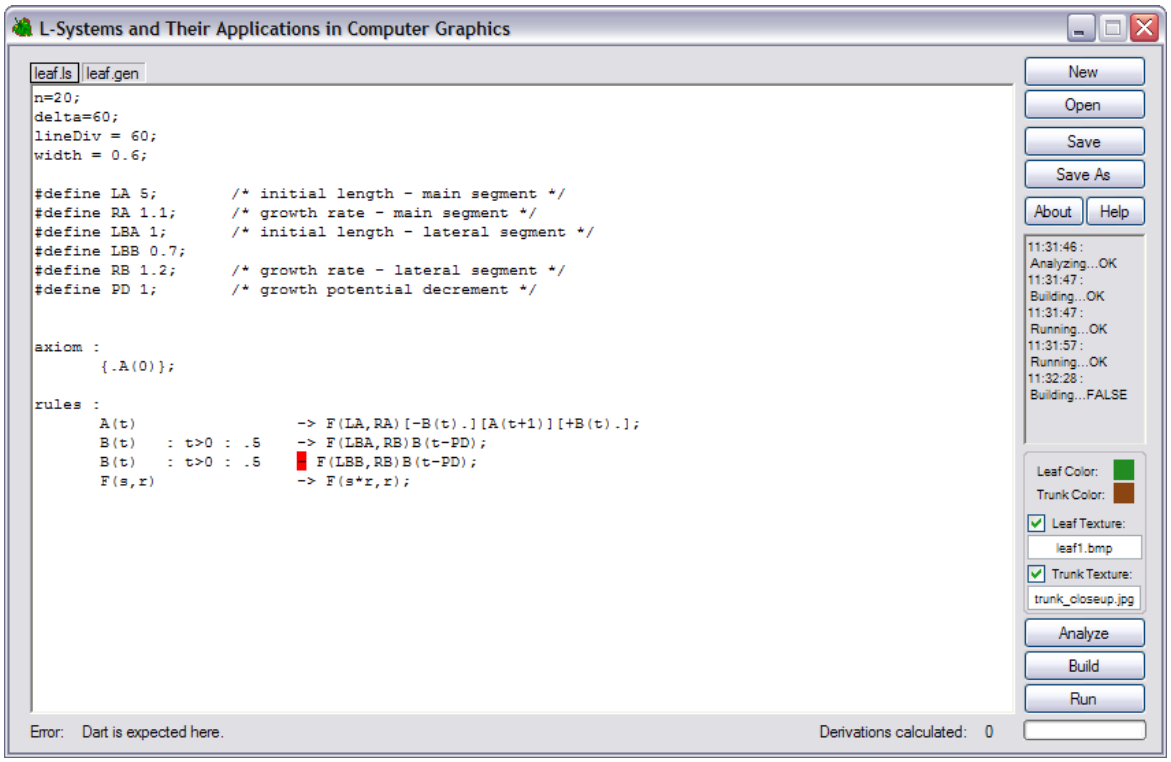
[18] Toman, J., Híšek. K. *Naší přírodou krok za krokem.* Albatros. 1994. 191 str. ISBN 80-00-00102-0

[19] Žára, J., Beneš, B., Felkel, P. *Modern Computer Graphics* (*Moderní počítačová grafika*). Computer Press, Brno, Czech Republic, 2004, in Czech.

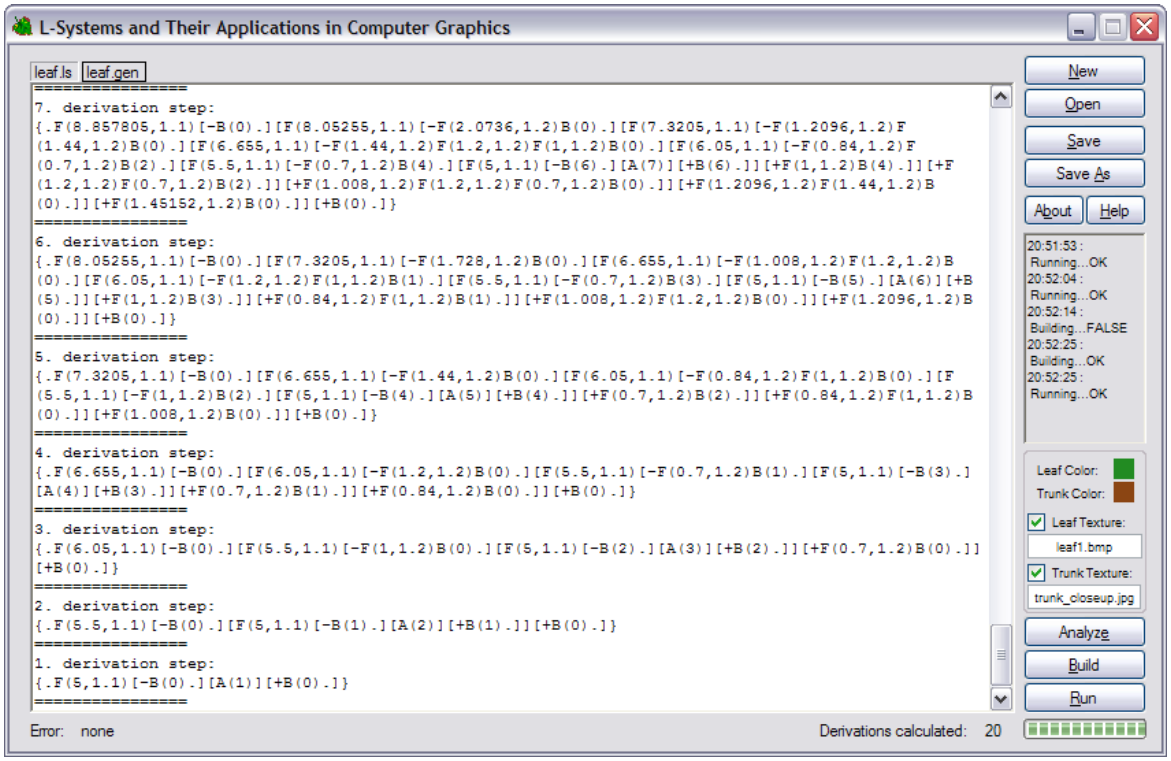
Seznam obrázků

Seznam příloh

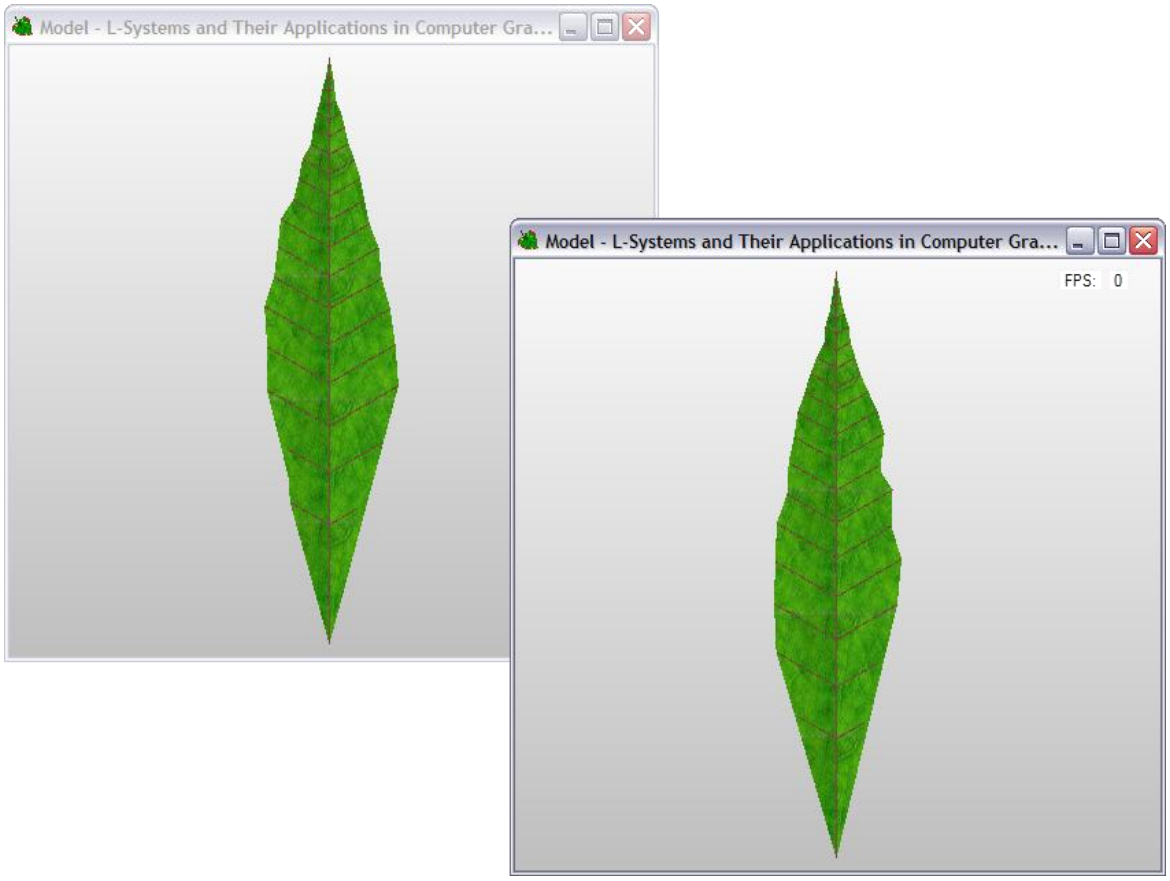
A. Uživatelské rozhraní interpretu



Obrázek A.1 Editační okno s vyznačenou syntaktickou chybou v kódu.



Obrázek A.2 Editační okno s vyobrazenými derivačními kroky.



Obrázek A.3 Interpretační okna s vyobrazenými stochastickými listy.

Interpretační okno se ovládá následovně:

- Rotace scény: *šipkami* klávesnice nebo posunem myši při stisknutém levém tlačítku.
- Posouvání scény: posunem myši při stisknutém pravém tlačítku
- Oddalování scény: klávesami *Page Down* a *Page Up* nebo kolečkem myši
- Uložení obrázku: klávesa *S*
- Zobrazení FPS: klávesa *F*
- Ovládání osvětlení: klávesa *L*
- Drátěný model: klávesa *D*
- Vypnutí šedého pozadí: klávesa *P*

B. Seznam symbolů interpretovaných želvou

- `F` : posunutí želvy vpřed s vykreslením úsečky.
- `f` : posunutí želvy vpřed bez vykreslení úsečky.
- `+` : rotace želvy doleva.
- `−` : rotace želvy doprava.
- `|` : otočení želvy.
- `&` : rotace želvy dolů.
- `^` : rotace želvy nahoru.
- `\` : rotace želvy na levý bok.
- `/` : rotace želvy na pravý bok.
- `[` : uložení pozice želvy na zásobník.
- `]` : načtení pozice ze zásobníku.
- `.` : označení vrcholu mnohoúhelníku.
- `{` : start mnohoúhelníku.
- `}` : konec mnohoúhelníku.
- `G` : posunutí želvy vpřed s vykreslením úsečky bez označení vrcholu mnohoúhelníku.
- `!` : zmenšení průměru segmentu.

C. Příklad syntaxe kódu pro interpret

Následující kód L-systému je napsán syntakticky tak, aby vyhovoval parseru mého interpretu. Je zde uvedena i hodnota podílu délky kroku želvy, aby se obrazec bez problému vešel do interpretačního okna. Pravidla přepsání modulů *A(t,a)* nikdy nemohou být aplikovaná stochasticky, protože jejich podmínky to logicky vylučují, ale přesto je u nich zadaná pravděpodobnost. Ta se sice nikdy nevyhodnotí, ale je zde uvedena pouze pro uspokojení parseru, který hlídá automaticky to, aby součet pravděpodobností pravidel se stejnou levou stranou byl vždy roven jedné.

```
n = 25;
delta = 60;
lineDiv = 250;
width = 0.6;

#define LA 5;           /* úvodní délka hlavního segmentu */
#define RA 1.15;        /* růst hlavního segmentu */
#define LB 1.3;         /* úvodní délka bočního segmentu */
#define RB 1.25;        /* růst bočního segmentu */
#define LC 3;           /* úvodní délka okrajového výčnělku */
#define RC 1.19;        /* růst okrajového výčnělku */

axiom : [{A(0,0).}][{A(0,1).}];

rules :
    A(t,a) : a==0 : .5   ->  F(LA,RA).[+B(t)F(LC,RC,t).}]
                             [+B(t){.]A(t+1,a);
    A(t,a) : a==1 : .5   ->  .F(LA,RA).[-B(t)F(LC,RC,t).}]
                             [-B(t){.]A(t+1,a);
    B(t)     : t>0       ->  F(LB,RB)B(t-1);
    F(s,r)               ->  F(s*r,r);
    F(s,r,t) : t>1       ->  F(s*r,r,t-1);
```

D. Obsah CD disku

CD:\

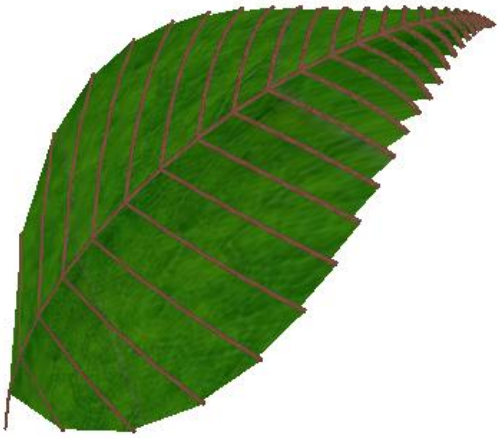
- aplikace
  - dokumentace - programová dokumentace
  - instalátor - instalační program aplikace
  - interpret - samostatný program bez instalátoru
  - zdrojové soubory - zdrojové soubory aplikace pro MS Visual Studio 2005
  - instalace.txt - popis instalace interpretu
  - požadavky.txt - popis požadavků aplikace
- písemná zpráva - písemná zpráva ve formátu pdf a doc
- příklady
  - obrázky modelů - obrázky L-systémů vygenerovaných interpretem
    - 2D - 2D modely
    - 3D - 3D modely
  - textury - textury použitelné pro modely
    - kmen - textury kmenů
    - list - textury listů
  - zdrojové soubory - zdrojové soubory L-systémů
    - 2D - 2D modely
    - 3D - 3D modely
- plakát.bmp - plakát demonstrující aplikaci

E. Výsledné modely

V této příloze jsou vyobrazeny další výsledné modely vygenerované pomocí mého interpretu stochastických 0L-systému, stejně jako všechny obrázky použité v této bakalářské práci, pokud tomu není uvedeno jinak. U následujících obrázků (kromě posledního) nejsou uvedeny zdrojové kódy těchto L-systémů, ale lze je nalézt na přiloženém CD disku.

E.1 List růže

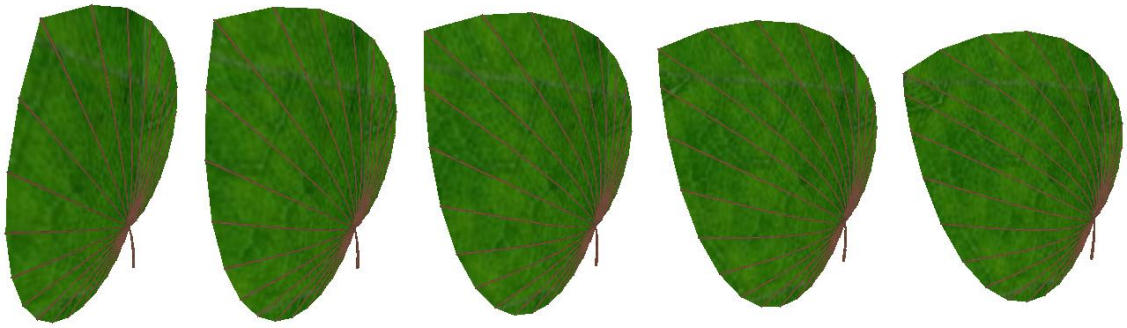
Zde je vyobrazen list růže, jenž je natočen do 3D prostoru vlivem gravitace.



Obrázek E.1 List růže s aplikovaným natočením v 3D prostoru.

E.2 Srdcovité listy

Obrázek E.2 vyobrazuje jednoduché srdcovité listy s aplikovaným různě velkým natočením čepele a na dalším obrázku jsou tyto listy aplikovány na jednoduchých stochastických rostlinkách, na kterých je náhodné postavení listu, velikost listu a natočení listu.



Obrázek E.2 Srdcovité listy s různým natočením čepele.

Obrázek E.3 Stochastické rostlinky tvořené srdcovitými listy.

E.3 Podlouhlé listy

Pro vygenerování těchto rostlinek se využilo jednoduchých podlouhlých listů (viz Kapitola 4.2). Jedná se o stochastické modely, u kterých je náhodné místo výskytu listů, velikost listů, zakřivení okraje listu a míra natočení listu vůči zemi. Velikost listů je ovšem i nadále omezena tak, aby se menší listy vyskytovali na rostlině výše než listy větší, neboli stejně tak, jak tomu je v přírodě. K interpretaci náhodné velikosti listu je využito derivačního zpoždění (viz Kapitola 4.3).

Obrázek E.4 Stochastické rostlinky s podlouhlými listy.

E.4 Plazivé rostlinky

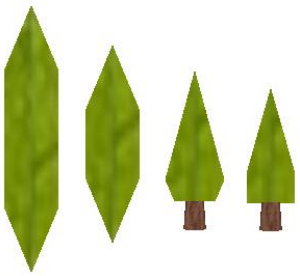
U těchto rostlin jsou listy generované vždy po pětici tak, aby tvořili jeden větší svazek neboli složený list. U rostlinky je náhodný růst segmentů, natočení svazku listů vůči zemi a natočení kmene. Na vrcholu jsou vždy nechané tři svazky stonků listů, jelikož počet derivačních kroků nebyl dostatečný pro vygenerování příslušných listů.

Obrázek E.5 Plazivé rostlinky se stochastickým růstem.

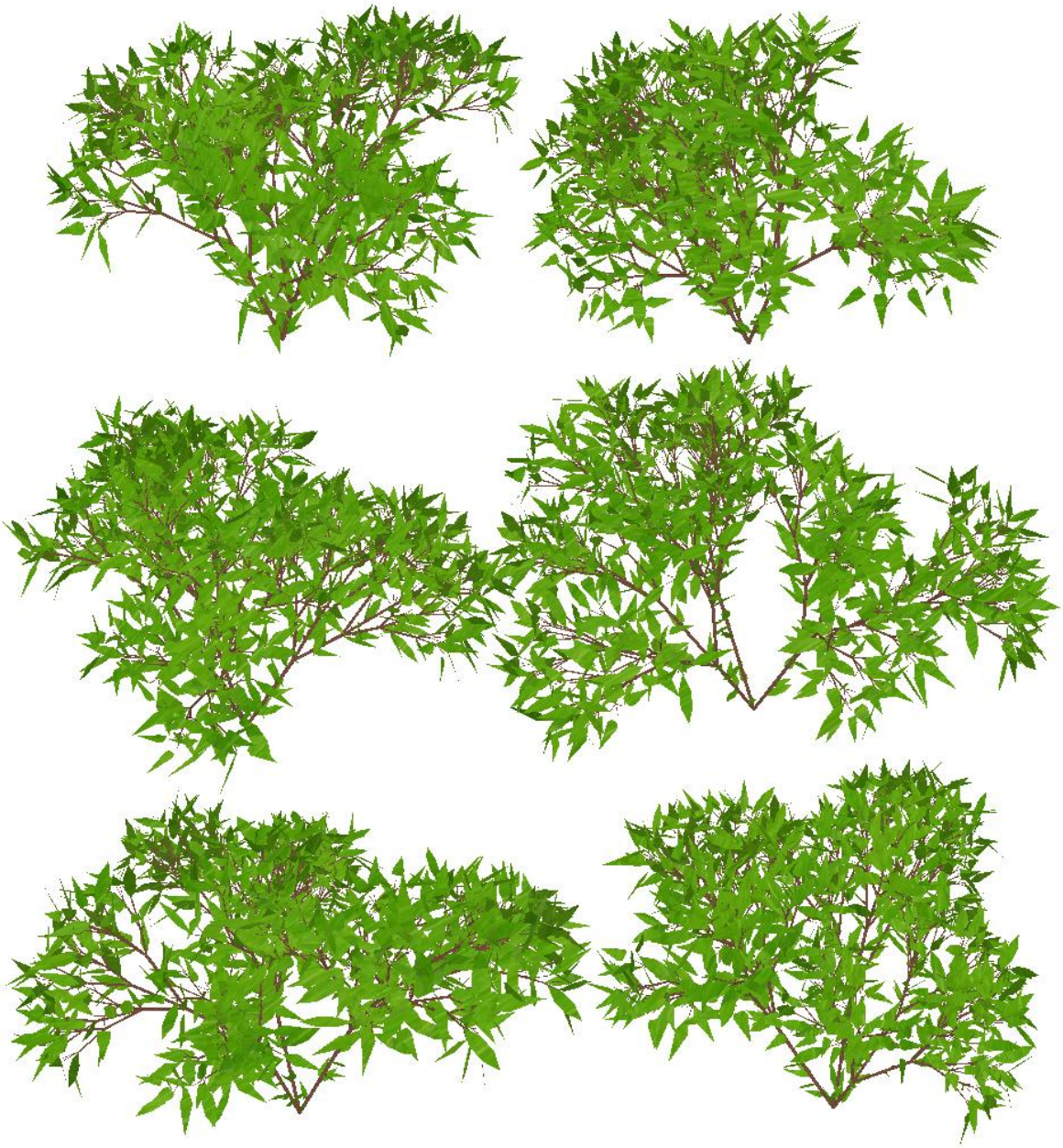
E.5 Keře

Následující kód stochastického 0L-systému je rozšířením pravidel (3.3) z kapitoly 3.3.2 a pomocí něho jsou vygenerované keře (viz Obrázek E.7). Symbol *A* provádí rozvětvení otcovského kmene na 3 dceřiné v různém natočení. Symbol *F* při každé derivaci vygeneruje pozici pro případný list (symbol *S*) a provede natočení svého segmentu, které ovlivní směr možného následujícího listu. Symbol *S* provede s pravděpodobností 84% prodloužení segmentu *F* (růst keře), s pravděpodobností 44% vygeneruje list (symbol *L*) a s pravděpodobností 28% dva listy. Dále symbol *L* vygeneruje náhodně jeden ze čtyř druhů listů (viz Obrázek E.6).

```
n = 7;
delta = 22.5;
lineDiv = 2;
width = 2;
widthDec = 0.35;

axiom : A;

rules :
    A : * : .7   -> [&FL!A]/////[&FL!A]///////[&FL!A];
    A : * : .3   -> [&FL!A]/////[&FL!A]\\\\\\\[&FL!A];
    F : * : .5   -> S ///// F;
    F : * : .5   -> S //////// F;
    S : * : .28  -> F L;
    S : * : .28  -> F;
    S : * : .28  -> F L \\ L;
    S : * : .16  -> L;
    L : * : .25  -> [^^{.-f.+ff.+f.-|-f.+ff.+f.}];
    L : * : .25  -> [^^{.-f.+f.+f.-|-f.+f.+f.}];
    L : * : .25  -> [^^{!G(.1)|f(.03)|[---f(.08).][--f(.15).]
                    [f(.5).][++f(.15).][+++f(.08).]}];
    L : * : .25  -> [^^{![G(.1)]f(.01)[--f(.12).]
                    [-f(.22).][f(.5).][+f(.22).][++f(.12).]}];
```



Obrázek E.6 Druhy listů vygenerované přepsáním symbolu L.



Obrázek E.7 Keře vygenerované jedním stochastickým 0L-systémem popsaným výše.