Matematicko - fyzikální fakulta
Univerzity Karlovy

DIPLOMOVÁ PRÁCE

Lindenmayerovy systémy v počítačové grafice

Vypracoval : Vítězslav Nový
Obor : Informatika

Vedoucí práce : RNDr. Josef Pelikán
Katedra software a výuky informatiky

Děkuji RNDr. Pelikánovi za cenné rady a připomínky k této práci.

Prohlašuji, že jsem celou práci vypracoval samostatně a že uvádím veškerou literaturu, kterou jsem při jejím zpracování používal.

V Praze dne 21. dubna 1995

Obsah

Úvod **1**

L-systémy a jejich využití v počítačové grafice **2**

Bezkontextové L-systémy (0L-systémy) 2
Geometrická interpretace řetězce pomocí želvičky 2
Interpretace řetězce pomocí 2D želvičky 2
Interpretace řetězce pomocí 3D želvičky 4
Větvené struktury 6
Osové stromy 6
0L-systémy nad osovými stromy 7
Závorkové 0L-systémy 7
Stochastické L-systémy 7
Kontextové L-systémy 8
Parametrické L-systémy 10
Parametrické 0L-systémy 10
Parametrické IL-systémy 12
Interpretace parametrických symbolů pomocí želvičky 13

Využití L-systémů při modelování vývoje rostlin **14**

Řídící mechanismy v rostlinách 14
Stochastický mechanismus 15
Vliv prostředí 15
Časový mechanismus 16
Mechanismus hromadění látek v růstovém vrcholu 16
Vývoj řízený signály 17
Modelování rostlinných orgánů 17
Vkládání předem definovaných ploch 17
Vyvíjející se modely 18
Modelování složených listů 19
Tropismus 20
Další prostředky pro modelování rostlin 21
Aplikace lsdraw 22

Uživatelský popis aplikace lsdraw **23**

Program rewrite 23
Formát vstupního souboru 24
Základní konstrukty 24
Používané konvence 25
Výrazy 25
Předefinovatelné symboly (ud-symboly) 26
Symboly řetězce 27
Klíčová slova 27
Struktura vstupního souboru 28
Sekce definice konstant 28
Sekce definice symbolů ignorovaných při testování kontextu 28
Sekce definice symbolů s default parametrem 28
Axiom 29
Definice tabulek pravidel 29
Definice přepisovacích pravidel 30
Vkládání souborů 30
Výstup z programu rewrite 30

Výstup v textovém formátu ... 31
Výstup v binárním formátu ... 31
Parametry programu ... 32
Program draw ... 33
Symboly interpretované želvičkou ... 34
Vkládání předdefinovaných objektů a textur ... 36
Vkládání předdefinovaných objektů ... 36
Vkládání textur ... 36
Zobrazení vygenerovaných objektů ... 37
Potlačení přímých linií ... 37
Modelování tropismu ... 37
Parametry programu draw ... 38
Program lsdraw ... 39
Uživatelské rozhraní ... 39
Zadávání parametrů pro program rewrite ... 40
Zadávání parametrů pro program draw ... 41
Parametry programu lsdraw ... 43
Dokumentace k aplikaci lsdraw ... 45
Sdílené moduly ... 45
Modul args ... 45
Modul darray ... 45
Modul errstr ... 45
Modul generr ... 45
Modul hash ... 45
Modul svalloc ... 46
Program rewrite ... 46
Fáze zpracování argumentů ... 46
Modul rarg ... 46
Fáze zpracování vstupního souboru ... 46
Modul lexical ... 46
Modul filestk ... 47
Modul parser ... 47
Modul tables ... 47
Modul product ... 48
Modul genexpr ... 48
Modul evaluate ... 48
Modul error ... 48
Fáze rozvoje řetězce ... 48
Modul param ... 48
Modul memstr ... 49
Modul deriv ... 51
Modul strio ... 51
Program draw ... 51
Využití knihovny Inventor v programu draw ... 51
Modul targ ... 53
Modul filetab ... 54
Modul t3d ... 54
Modul viewer ... 56
Uživatelské rozhraní aplikace - program lsdraw ... 56
Modul ui ... 56
Modul data ... 56
Modul cmdline ... 56
Některá možná rozšíření aplikace ... 57
Závěr ... 58

Literatura **59**

Úvod

Lindenmayerovy systémy (zkráceně L-systémy) definoval a poprvé použil v roce 1968 Aristid Lindenmayer ve své práci o modelování vývoje jednoduchých vícebuněčných organismů [1]. Později byly L-systémy dále rozvíjeny pro účely modelování stále složitějších struktur. Byly definovány kontextové, stochastické, parametrické a další třídy L-systémů, které mohou být využity pro generování fraktálních útvarů, soběpodobných struktur, křivek vyplňujících celou rovinu, různých uměleckých vzorů, modelování vývoje biologických struktur od kolonií buněk po vyšší rostliny, atd.

L-systémy lze zjednodušeně popsat jako gramatiky, u kterých se v jednom derivačním kroku přepisují všechny symboly řetězce současně. Použitím L-systému tedy získáme řetězec symbolů, který je třeba nějakým způsobem interpretovat. Při využití L-systémů v počítačové grafice je tedy třeba věnovat pozornost nejen samotným L-systémům, ale i metodám grafické interpretace vygenerovaných řetězců. Pro účely modelování vývoje rostlin byly nalezeny takové metody interpretace řetězce, které umožňují vytvářet i realistické obrazy modelovaných rostlin.

Při modelování rostlin pro účely počítačové grafiky se ve velké míře využívají biologické znalosti o jejich vývoji a mechanismech, které ho řídí. L-systémy a jejich grafická interpretace dávají na druhé straně velké možnosti biologům. Umožňují jim sledovat vývoj rostliny za různých podmínek, testovat teorie o vnitřních mechanismech řídících vývoj rostliny i o působení vnějšího prostředí. Nabízejí prostředky pro zjišťování parametrů těchto mechanismů jako je např. rychlost šíření signálů v rostlině apod.

Cílem této práce bylo vytvořit program, který je schopen pracovat s různými třídami L-systémů, generovat jimi popsané struktury a ty potom zobrazovat.

V kapitole 1 je uveden přehled různých tříd L-systémů a základní metody interpretace řetězců generovaných těmito L-systémy.

Kapitola 2 je zaměřena na využití L-systémů při modelování vývoje rostlin. Jsou zde diskutovány některé problémy specifické pro modelování rostlin. Dále jsou zde uvedena některá rozšíření metody interpretace řetězce, která umožňují vytvářet realistické obrazy rostlin.

Kapitola 3 obsahuje uživatelský popis aplikace lsdraw. Tato aplikace zahrnuje funkce potřebné pro modelování vývoje rostlin od načtení L-systému zadaného v textovém formátu až po možnost animace vygenerovaných vývojových sekvencí.

Kapitola 4 obsahuje stručnou programátorskou dokumentaci k aplikaci lsdraw.

Kapitola 1

L-systémy a jejich využití v počítačové grafice

Bezkontextové L-systémy (0L-systémy)

Definice 1.1
Nechť V je abeceda, V^* je množina slov nad abecedou V a V^+ je množina neprázdných slov nad abecedou V. *Řetězcový 0L-systém* je uspořádaná trojice $G = \langle V, \omega, P \rangle$, kde V je *abeceda* systému, $\omega \in V^+$ je neprázdné slovo nazývané *axiom* a $P \subset V \times V^*$ je konečná množina *přepisovacích pravidel*. Pravidlo $(a, \chi) \in P$ budeme také zapisovat ve tvaru $a \to \chi$. Znak a se nazývá *přepisovaný symbol* a řetězec χ se nazývá *pravá strana* pravidla. Předpokládá se, že pro každé $a \in V$ existuje $\chi \in V^*$ takové, že $a \to \chi$. Pokud není pro některý přepisovaný symbol $a \in V$ explicitně zadáno žádné pravidlo, předpokládá se, že P obsahuje *identické pravidlo* $a \to a$. 0L-systém se nazývá *deterministický* (zkráceně DOL-systém), jestliže pro každé $a \in V$ existuje právě jedno $\chi \in V^*$ takové, že $(a, \chi) \in P$.

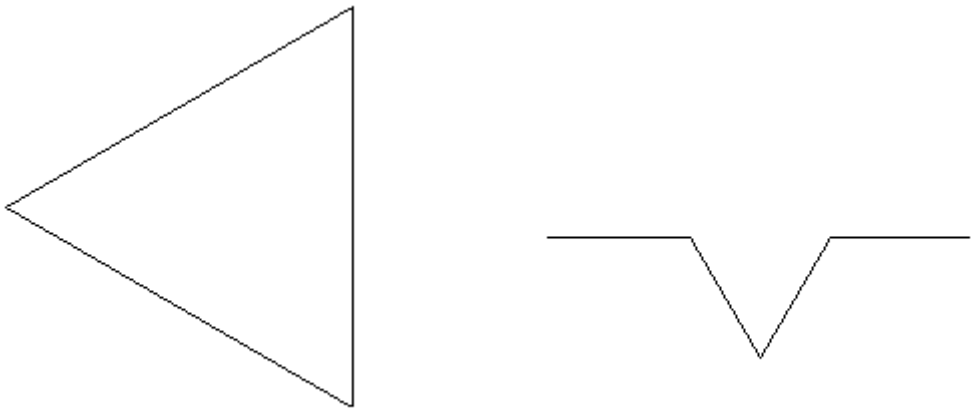
Definice 1.2
Nechť $\mu = a_1 \dots a_m$ je libovolné slovo nad abecedou V. Řekneme, že slovo $\nu = \chi_1 \dots \chi_m \in V^*$ je *přímo odvozeno* ze slova μ (nebo také *generováno* slovem μ) a značíme $\mu \Rightarrow \nu$, právě když $a_i \to \chi_i$ pro všechna $i = 1 \dots m$. Slovo ν je generováno L-systémem G *derivací délky* n, právě když existuje taková *derivační posloupnost* slov $\mu_0, \mu_1, \dots \mu_n$, že $\mu_0 = \omega$, $\mu_n = \nu$ a $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$.

Geometrická interpretace řetězce pomocí želvičky

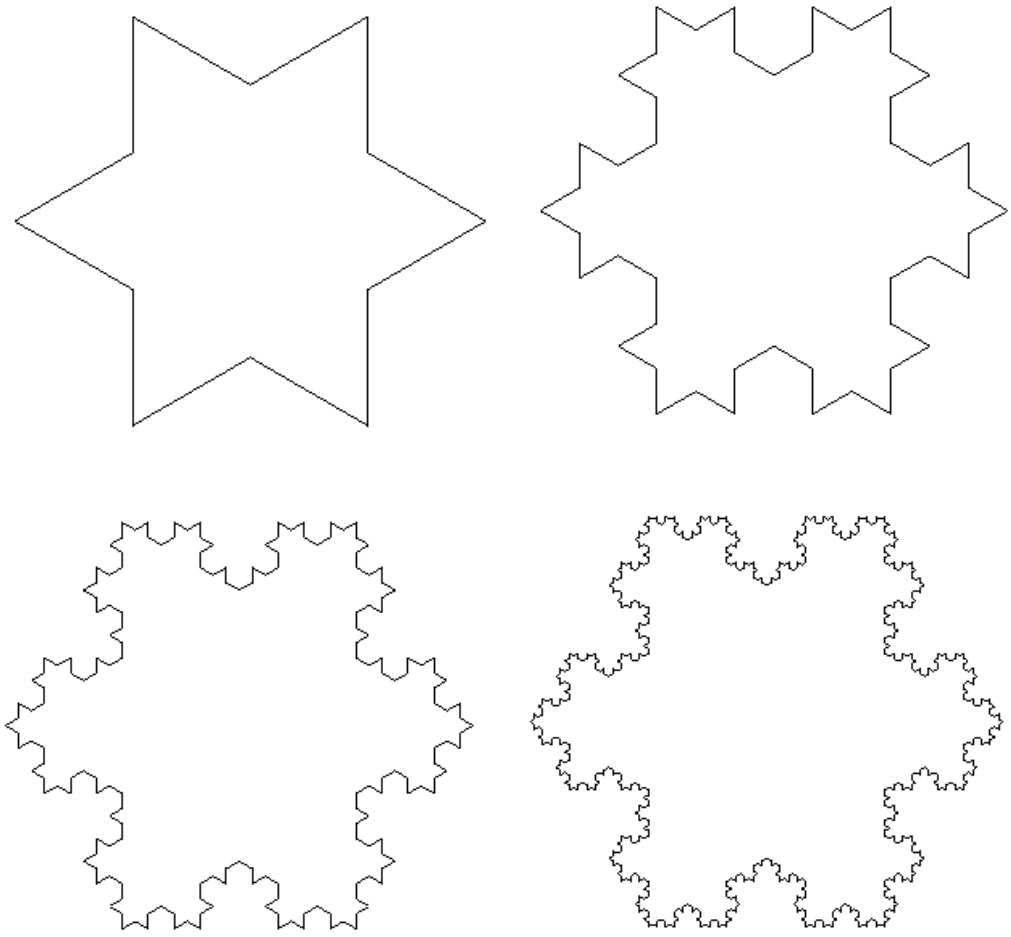
Základní myšlenkou využití L-systémů v počítačové grafice je generování řetězců a jejich následná geometrická interpretace. Jednou z možností jak vygenerovaný řetězec interpretovat je využití želvičky známé z jazyka LOGO. Řetězec je v tomto případě považován za posloupnost příkazů pro tuto želvičku. Postupným vykonáváním příkazů želvička vytváří obrazec reprezentovaný vygenerovaným řetězcem. V následujícím textu popíšeme podrobněji dva typy želviček generujících výsledný obrazec v rovině resp. v prostoru.

Interpretace řetězce pomocí 2D želvičky

Stav 2D želvičky je definován jako uspořádaná trojice (x, y, α), kde uspořádaná dvojice (x, y) určuje kartézské souřadnice *pozice* želvičky a úhel α určuje směr dalšího pohybu želvičky. Dále je zadána *délka kroku* d a *změna úhlu* δ.



Obr. 1.1 Axiom a útvar nahrazující přepisovanou hranu při generování fraktálního útvaru nazývaného vločka Kochové



Obr. 1.2 První čtyři kroky vývoje vločky Kochové generované L-systémem (1.1)

Želvička interpretuje příkazy reprezentované následujícími symboly:

F Želvička se posune vpřed o krok délky *d*. Její stav se změní z *(x, y,* α*)* na *(x', y',* α*)*, kde $x' = x + d \cos \alpha$ a $y' = y + d \sin \alpha$. Nakreslí úsečku určenou body *(x, y)* a *(x', y')*.

f Želvička se posune vpřed o krok délky *d* aniž by nakreslila odpovídající úsečku.

\+ Želvička se otočí doleva o úhel δ. Její stav se změní z *(x, y,* α *)* na *(x, y,* $\alpha + \delta$ *)*.

\- Želvička se otočí doprava o úhel δ. Její stav se změní z *(x, y,* α *)* na *(x, y,* $\alpha - \delta$ *)*.

Ostatní symboly v interpretovaném řetězci jsou ignorovány.

Nechť je dán řetězec ν, počáteční stav želvičky *(x, y,* α *)*, a konstantní parametry *d* a δ. *Interpretace řetězce* ν *2D želvičkou* je obrazec (množina úseček), nakreslený 2D želvičkou podle příkazů v řetězci ν.

Známý fraktální útvar - vločku Kochové - lze generovat (s konečnou hloubkou rekurze) L-systémem

$$\omega : F + + F + + F \qquad \textbf{(1.1)}$$
$$F \rightarrow F - F + + F - F$$

kde $\delta = 60^{o}$. Na obrázku (1.1) je útvar vygenerovaný 2D želvičkou interpretací axiomu L-systému (1.1) a útvar, kterým se v každém derivačním kroku nahrazuje každá hrana obrazce (je to útvar generovaný 2D želvičkou interpretací pravé strany pravidla L-systému). Obrázek (1.2) pak ukazuje vločku Kochové v prvních čtyřech krocích vývoje.

Interpretace řetězce pomocí 3D želvičky

Model 2D želvičky lze rozšířit následujícím způsobem na 3D model. Stav 3D želvičky je definován *pozicí a orientací*. Pozice je stejně jako u 2D modelu určena kartézskými

souřadnicemi. Orientace je pak dána třemi vektory **H**, **L**, **U**, které označují směr vpřed, vlevo a nahoru. Tyto vektory mají jednotkovou délku, jsou navzájem kolmé a splňují rovnost $\mathbf{H} \times \mathbf{L} = \mathbf{U}$.

Otočení želvičky pak můžeme vyjádřit předpisem

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

kde **R** je příslušná rotační matice o velikosti 3×3. Speciálně, rotace želvičky o úhel α kolem osy určené vektorem **U**, **L**, resp. **H** je reprezentována maticemi:

$$\mathbf{R}_{\mathbf{U}}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_{\mathbf{L}}(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$\mathbf{R}_{\mathbf{H}}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Orientace 3D želvičky v prostoru je řízena příkazy, která jsou reprezentovány následujícími symboly:

\+ Rotace doleva o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{U}}(\delta)$.

\- Rotace doprava o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{U}}(-\delta)$.

& Rotace dolů o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{L}}(-\delta)$.

^ Rotace nahoru o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{L}}(\delta)$.

\ Nahnutí želvičky doleva o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{H}}(\delta)$.

/ Nahnutí želvičky doprava o úhel δ. Použije se rotační matice $\mathbf{R}_{\mathbf{H}}(-\delta)$.

| Otočení želvičky o 180°. Použije se rotační matice $\mathbf{R}_U(180°)$.

Větvené struktury

Výsledkem interpretace řetězce pomocí želvičky je vždy lomená čára. Některé její segmenty se mohou protínat, mohou se překrývat nebo být neviditelné, ale vždy se jedná o lomenou čáru.

Chceme-li tedy použít tuto metodu pro generování obrázků rostlin, je třeba ji rozšířit o možnost vytvářet obecné stromové struktury. Jako základ pro další úvahy nám poslouží kořenové stromy z teorie grafů, rozšířené o botanicky motivovanou představu osy větve.

Osové stromy

Definice 1.3
Kořenový strom je orientovaný strom, jehož hrany jsou označeny identifikátory. V kořenovém stromě je označen jeden význačný vrchol zvaný *kořen*. Vrchol, který není kořenem, a je obsažen pouze v jedné hraně se nazývá *koncový* vrchol. Hrany jsou orientovány tak, že tvoří cesty z kořene do koncových vrcholů. V biologickém kontextu odpovídají hrany segmentům větví. Segment za kterým následuje v cestě od kořene ke koncovému vrcholu alespoň jeden další segment s nazývá *vnitřní segment*. Koncový segment (za kterým nenásleduje další segment) se nazývá *vrchol*.

Definice 1.4
Osový strom je speciálním typem kořenového stromu. Mezi hranami vycházejícími z každého vrcholu je označena vždy nejvýše jedna hrana jako *přímý segment*. Ostatní hrany jsou označeny jako *vedlejší segment.* Cesta v osovém stromě se nazývá *osa* jestliže:

- první segment začíná v kořeni nebo je to vedlejší segment nějakého uzlu,
- každý následující segment je přímý segment a
- poslední segment cesty není ve stromě následován žádným přímým segmentem.

Každá osa se všemi svými následníky tvoří *větev*. Větev je sama o sobě osovým stromem.

Osy a větve jsou číslovány podle úrovní, na kterých začínají. Osa začínající v kořeni je řádu 0. Osa začínající jako vedlejší segment osy řádu n je řádu $n + 1$. Řád větve je určen jako minimum z řádů os, které větev obsahuje.

0L-systémy nad osovými stromy

Vzhledem k tomu, že naším cílem je modelovat pomocí L-systémů vývoj rostlin, můžeme použít přepisovací mechanismus L-systémů přímo na osové stromy, které mohou velmi dobře reprezentovat jejich topologickou strukturu.

Přepisovací pravidla těchto systémů nahrazují hranu osovým stromem tak, že kořen je připojen místo počátečního vrcholu a koncový vrchol osy řádu 0 místo koncového vrcholu přepisované hrany. 0L-systém nad osovými stromy je určen třemi komponentami: množinou symbolů označujících hrany V, počátečním stromem ω a množinou přepisovacích pravidel P.

Definice 1.5
Nechť G je L-systém nad osovými stromy. Říkáme, že osový strom T_2 je generován osovým stromem T_1 a píšeme $T_1 \Rightarrow T_2$, jestliže lze strom T_2 získat paralelním nahrazením všech hran stromu T_1 v souladu s pravidly z množiny P. Strom T je generován systémem G derivací délky n, jestliže existuje posloupnost stromů $T_0, T_1, \ldots T_n$ taková, že $T_0 = \omega$, $T_n = T$ a $T_0 \Rightarrow T_1 \Rightarrow \ldots \Rightarrow T_n$.

Závorkové 0L-systémy

Definice L-systémů nad osovými stromy neřeší problém reprezentace stromů datovými strukturami. Jednou z možností jak tento problém řešit je stromová struktura vytvořená pomocí ukazatelů. Další možností je reprezentovat strom pomocí *řetězce se závorkami.* V řetězci reprezentujícím danou větev je řetězec reprezentující větev vyššího řádu uzavřen v hranatých závorkách.

V souladu s touto reprezentací stromu rozšíříme množinu symbolů interpretovaných želvičkou o následující příkazy:

[Uloží aktuální stav želvičky na zásobník. Uložená informace obsahuje pozici a orientaci želvičky a případně další doplňující informace jako barvu a tloušťku kreslené čáry.

] Vybere ze zásobníku stav želvičky a uvede ji do tohoto stavu. Není kreslena žádná čára, ačkoli obecně se poloha želvičky změní.

Stochastické L-systémy

V případě, že při generování dvou rostlin použijeme tentýž deterministický L-systém, dostaneme jako výsledek dvě identické rostliny. To není dobré, chceme-li do jednoho obrázku generovat více rostlin stejného druhu. Ani při generování jedné rostliny nepůsobí dobře pravidelné opakování stejných struktur. Vložením náhodných prvků do interpretace

řetězce (tzv. randomizací interpretace) lze tyto problémy částečně vyřešit. Tímto způsobem lze ovlivnit například délku segmentů nebo velikosti úhlů mezi nimi, ale ne topologii rostliny. Řešení tohoto problému nabízejí právě stochastické L-systémy.

Definice 1.6
Stochastický L-systém je uspořádaná čtveřice $G_\pi = (V, \omega, P, \pi)$. Abeceda V axiom ω a množina přepisovacích pravidel P mají stejný význam jako ve výše uvedené definici 0L-systému (definice 1.1). Funkce $\pi : P \rightarrow (0, 1 \rangle$, nazývaná *rozdělení pravděpodobnosti*, zobrazuje množinu přepisovacích pravidel do množiny *pravděpodobností pravidel*. Funkce π musí mít tu vlastnost, že součet pravděpodobností všech pravidel se stejnou levou stranou je roven 1.

Definice 1.7
Odvození $\mu \Rightarrow \nu$ nazýváme *stochastické odvození* v G_π, jestliže pro každý výskyt znaku a ve slově μ je pravděpodobnost aplikace pravidla $p = (a, \chi)$ na tento znak rovna $\pi(p)$. To tedy znamená, že každý výskyt znaku a v daném slově může být v jednom derivačním kroku nahrazen v podle jiného pravidla tvaru (a, χ).

Pravidla stochastických L-systémů budeme zapisovat ve tvaru:

$$a \rightarrow (\pi (p)) \chi ,$$

čili stejným způsobem jako pravidla OL-systémů, ale navíc za šipkou oddělující levou a pravou stranu napíšeme do závorky hodnotu funkce π pro toto pravidlo, tj. pravděpodobnost aplikace toho pravidla na symbol vysktující se na levé straně pravidla.

Kontextové L-systémy

Pravidla v 0L-systémech jsou bezkontextová, tj. jejich použití nezávisí na kontextu ve kterém se přepisovaný symbol nachází. Použití kontextových pravidel však může přinést nové možnosti například při modelování šíření různých látek v rostlině.

Kontextové L-systémy můžeme rozdělit podle tvaru pravidel do několika tříd. *2L-systémy* obsahují pravidla ve tvaru $a_l < a > a_r \rightarrow \chi$, kde znak a (přepisovaný symbol) může být nahrazen slovem χ, právě když se před ním v řetězci nachází znak a_l a následuje za ním znak a_r. Pravidla v *1L-systémech* obsahují buď pouze levý nebo pouze pravý kontext, tj. jsou tvaru $a_l < a \rightarrow \chi$ nebo $a > a_r \rightarrow \chi$. 0L-systémy, 1L-systémy a 2L-systémy spadají do širší třídy *IL-systémů*, zvaných též *(k, l)-systémy*. V (k, l)-systémech je levý kontext slovo délky k a pravý kontext slovo délky l. Z důvodu zjednodušení, upravíme definici IL-systémů tak, že povolíme existenci pravidel s kontexty různých délek v jednom systému. Kromě toho stanovíme, že aplikace kontextového pravidla má přednost před aplikací bezkontextového pravidla se stejným přepisovaným symbolem. Jestliže je tedy možné nahradit daný znak podle kontextového i bezkontextového pravidla, bude použito

kontextové pravidlo. Pokud nebylo pro nějaký znak zadáno žádné přepisovací pravidlo budeme jako v předchozích případech uvažovat existenci identického pravidla.

Následující L-systém je příkladem použití kontextových pravidel pro modelování šíření signálu.

$$\omega : baaaaaaa \qquad \textbf{(1.2)}$$
$$p_1: b < a \rightarrow b$$
$$p_2: b \rightarrow a$$

Několik prvních slov vygenerovaných tímto systémem bude vypadat takto:

baaaaaaa
abaaaaaa
aabaaaaa
aaabaaaa
aaaabaaa
...

Znak *b* se pohybuje z levé strany řetězce na pravou.

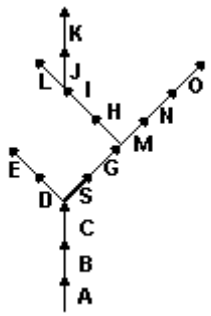
V kontextových L-systémech nad osovými stromy se pravá strana pravidla skládá ze tří částí: *cest l* a *p*, určujících levý a pravý kontext a *přepisované hrany a.* Hranu označenou znakem *a* je možné přepsat podle pravidla tvaru $\gamma_l < a > \gamma_r \rightarrow \chi$, právě když γ_l je cesta v osovém stromě, končící v počátečním vrcholu hrany *a* a γ_r je cesta začínající v koncovém vrcholu hrany *a.* Z definice osového stromu je zřejmé, že zatímco existuje pouze jedna cesta délky $|\gamma_l|$, která vede do počátečního vrcholu přepisované hrany, a je tedy třeba ji testovat na rovnost s levým kontextem pravidla, kandidátů na rovnost s pravým kontextem pravidla, tj. cest délky $|\gamma_r|$ začínajících v koncovém vrcholu přepisované hrany může být více. V některých pracích se však testování pravého kontextu omezuje pouze na cestu, která začíná v koncovém vrcholu nahrazované hrany a skládá se pouze z přímých segmentů.

Použijeme-li pro reprezentaci osových stromů uzávorkované řetězce, bude testování kontextu o něco složitější, protože dvě sousední hrany v osovém stromě mohou být v řetězci odděleny symboly reprezentujícími větve vyššího řádu. Například z obrázku (1.3) plyne, že pravidlo s levou staranou BC < F > GM může být aplikováno na symbol S v řetězci

ABC[DE][SG[HI[JK]L]MNO] **(1.3)**

což znamená, že při testování levého kontextu symbolu F byla vynechána větev vyššího řádu [DE] a při testování pravého kontextu větev [JK]. Je také zřejmé, že při testování

levého kontextu se větve vyššího řádu vždy vynechávají, zatímco při testování pravého kontextu je nutno otestovat jak osu, tak postupně projít všechny větve vyššího řádu.



Obr. (1.3) Schéma osového stromu reprezentovaného řetězcem (1.3)

Vzhledem k tomu, že rozvíjení řetězce obvykle probíhá spíše v závislosti na topologii než na geometrii obrazce, je nutné neuvažovat některé symboly při testování kontextu. To se obvykle zajišťuje definováním množiny symbolů, které systém při testování kontextu ignoruje.

Parametrické L-systémy

Ačkoli výše uvedené třídy L-systémů umožňují prostřednictvím interpretace pomocí želvičky generovat velkou třídu objektů od fraktálních útvarů po rostlinám podobné větvené struktury, jsou jejich možnosti přesto značně omezené. Délka každé čáry ve výsledném obrazci musí být celočíselným násobkem délky segmentu. Jestliže tedy chceme vytvořit obrazec, který obsahuje různě dlouhé úsečky, může se stát, že délka segmentu, což je (v nejlepším případě největší) společný dělitel délek všech úseček v obrazci, bude velmi malá. Obrazec se pak bude skládat z velkého množství segmentů, což může negativně ovlivnit jak nároky na paměť, tak rychlost výpočtu a zobrazování. Toto omezení také znemožňuje reprezentovat přesně některé tvary, jako například rovnoramenný pravoúhlý trojúhelník. Vzhledem k existenci nejkratší možné úsečky, nelze tento útvar někdy reprezentovat ani přibližně s dostatečnou přesností. Obdobné argumenty platí i pro úhly mezi úsečkami. Pro vyřešení podobného problému navrhl Lindenmayer v [2] připojit ke znakům abecedy L-systému číselné parametry. Definici parametrických L-systémů formulovali později Prusinkiewicz a Hanan v [4].

Parametrické 0L-systémy

Parametrické L-systémy pracují s *parametrickými řetězci*, což jsou řetězce symbolů, skládajících se ze *znaků abecedy* a jim přiřazených parametrů. Znaky jsou prvky *abecedy V* a parametry jsou reálná čísla. Symbol skládající se ze znaku A a parametrů $a_1, a_2, \ldots a_n \in$ R budeme označovat $A (a_1, a_2, \ldots a_n)$. Symboly jsou prvky množiny $M = V \times R^*$, kde R je množina všech konečných posloupností parametrů. Množiny všech řetězců symbolů a

všech neprázdných řetězců symbolů budeme označovat $M^* = (V \times R^*)^*$ resp. $M^+ = (V \times R^*)^+$.

Aktuální parametry, které se nacházejí v přepisovaném slově korespondují s *formálními parametry*, definovanými v přepisovacích pravidlech parametrického L-systému. Nechť Σ je množina formálních parametrů, pak C(Σ) označuje *logický výraz* s parametry z množiny Σ a E(Σ) označuje *aritmetický výraz* s parametry z množiny Σ. Oba typy výrazů se skládají z formálních parametrů a číselných konstant kombinovaných s aritmetickými operátory +, - , *, / , ^ (sčítání, odčítání, násobení, dělení a mocnina), relačními operátory <, >, = (větší než, menší než a rovno), logickými operátory !, &, | (negace, logický součin a logický součet) a závorkami. Jsou zachována standardní pravidla pro tvorbu syntakticky správných výrazů a standardní priority operátorů. Logický výraz určený prázdným řetězcem má hodnotu 1. Množinu správně zkonstruovaných logických výrazů s parametry z množiny Σ budeme označovat $C(\Sigma)$ a množinu všech správně zkonstruovaných aritmetických výrazů s parametry z Σ budeme značit $E(\Sigma)$. Dále označíme množinu všech konečných řetězců s prvky z množiny $E(\Sigma)$ symbolem $E(\Sigma)^*$.

Definice 1.8
Parametrický 0L-systém je pak definován jako uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde

- V je *abeceda* systému,
- Σ je množina *formálních parametrů,*
- $\omega \in (V \times R^*)^+$ je neprázdné parametrické slovo nazývané axiom,
- $P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times E(\Sigma)^*)^*$ je konečná množina pravidel.

Symboly : a $\rightarrow$ použijeme k oddělení tří částí pravidla: *přepisovaného symbolu, podmínkového výrazu* a *pravé strany pravidla.* Například pravidlo s přepisovaným symbolem $A(t)$, podmínkou $t > 5$ a pravou stranou $B(t + 1)C(t \wedge 0.5, t - 1)$ zapíšeme následujícím způsobem:

$$A(t) : t > 5 \rightarrow B(t + 1)\, C(t \wedge 0.5, t - 1) \qquad \textbf{(1.4)}$$

Přepisovací pravidlo může být aplikováno na symbol v parametrickém řetězci, jestliže jsou splněny následující podmínky:

- znak v přepisovaném symbolu v pravidle je týž, jako znak v nahrazovaném symbolu v řetězci,

- nahrazovaný symbol má stejný počet skutečných parametrů, jako přepisovaný symbol v pravidle formálních parametrů,

- dosadíme-li do pravidla za formální parametry skutečné parametry z nahrazovaného symbolu, bude podmínkový výraz pravdivý.

Aplikujeme-li pravidlo s přepisovaným symbolem $A (x_1, ..., x_n)$ na symbol $A (p_1, ..., p_n)$, je tento symbol v řetězci nahrazen pravou stranou aplikovaného pravidla, v níž byl každý výskyt formálního parametru x_i nahrazen skutečným parametrem p_i pro všechna $i = 1 ... n$. Například pravidlo (1.4) lze aplikovat na symbol $A (9)$, protože tento symbol má jeden aktuální parametr a pravidlo jeden formální parametr a podmínkový výraz $t > 5$ je pro $t = 9$ pravdivý. Výsledkem aplikace tohoto pravidla bude parametrické slovo $B (10) C (3, 8)$.

Definice 1.9
Jestliže parametrické slovo χ lze získat aplikací nějakého pravidla na symbol a v parametrickém L-systému G, píšeme $a \approx> \chi$. Nechť $\mu = a_1 ... a_m$ je parametrické slovo. Řekneme, že slovo $\nu = \chi_1 ... \chi_m \in V^*$ je *přímo odvozeno* ze slova μ (nebo také *generováno* slovem μ) a značíme $\mu \Rightarrow \nu$, právě když $a_i \approx> \chi_i$ pro všechna $i = 1 ... m$. Parametrické slovo ν je generováno L-systémem G *derivací délky n*, právě když existuje posloupnost slov $\mu_o, \mu_1, ... \mu_n$ taková, že $\mu_0 = \omega$, $\mu_n = \nu$ a $\mu_0 \Rightarrow \mu_1 \Rightarrow ... \Rightarrow \mu_n$.

Parametrické IL-systémy

Parametrický IL-sytsém je uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde symboly V, Σ a ω mají stejný význam, jako u parametrických 0L-systémů. P je opět množina přepisovacích pravidel. U parametrických IL-systémů se levá strana pravidla skládá z levého kontextu, přepisovaného symbolu a pravého kontextu. Levý a pravý kontext jsou parametrické řetězce se znaky z abecedy V a parametry z množiny Σ. Všechny parametry, které se nacházejí na levé straně pravidla, tj. v přepisovaném symbolu a levém i pravém kontextu, se pak mohou vyskytovat v podmínkovém výrazu i ve výrazech v pravé straně pravidla.

Například pravidlo

$$A (x) B (y, z) < C (w) > D (u, v) : u + v + x + y + z < 20 \rightarrow E (x + y, 3 * z) F (v - u) G (2 * x, z, w) \quad \textbf{(1.5)}$$

může být aplikováno na symbol $C (7)$, který se v parametrickém řetězci nachází v kontextu

$$A (4) B (4, 5) C (7) D (1, 2) \quad \textbf{(1.6)}$$

Symbol $C (7)$ bude při této operaci nahrazen parametrickým řetězcem

$$E (8, 15) F (1) G (8, 5, 7). \quad \textbf{(1.7)}$$

Interpretace parametrických symbolů pomocí želvičky

Symboly s parametry nám při řízení želvičky dávají oproti symbolům bez parametrů nové možnosti. Má-li řídící symbol alespoň jeden parametr, bude jeho první parametr použit při provádění příslušné akce. Pokud řídící symbol žádné parametry nemá, bude stejně jako v případě neparametrických systémů použita default hodnota. Následující tabulka obsahuje základní sadu parametrických příkazů:

F (a) Želvička se posune vpřed o krok délky *a.* Souřadnice její pozice se změní z *(x, y, z)* na *(x', y', z')*, kde $x' = x + a\mathbf{H}_x$, $y' = y + a\mathbf{H}_y$ a $z' = z + a\mathbf{H}_z$. Nakreslí úsečku danou body *(x, y, z), (x', y', z').*

f (a) Želvička se posune vpřed o krok délky *a.* Úsečka mezi původní a novou pozicí nebude vykreslena.

+ (a) Želvička se otočí o *a* stupňů kolem osy **U.** Jestliže je *a* kladné, otočí se doleva. Jestliže je *a* záporné otočí se doprava.

& (a) Želvička se otočí o *a* stupňů kolem osy **L.** Jestliže je *a* kladné, otočí se nahoru. Jestliže je *a* záporné otočí se dolů.

/ (a) Želvička se otočí o *a* stupňů kolem osy **H.** Jestliže je *a* kladné, nahne se doprava. Jestliže je *a* záporné nahne se doleva.

Příkazy reprezentované symboly -, ^ a \ v tabulce neuvádíme, protože každý z nich lze nahradit inverzním příkazem s parametrem s opačným znaménkem. Jejich význam je však vhodné zachovat kvůli kompatibilitě s neparametrickými systémy.

Poznamenejme, že symboly +, /, ^ a & jsou používány jednak jako znaky abecedy systému a jednak jako operátory ve výrazech. Jejich význam je vždy zřejmý z kontextu.

Kapitola 2

Využití L-systémů při modelování vývoje rostlin

V této kapitole se zaměříme na modelování vývoje bylin, tedy nedřevěných rostlin, jejichž vývoj je ovlivněn především vnitřními řídícími mechanismy. Vývoj stromů a jiných dřevěných rostlin je naopak z velké části ovlivněn prostředím, konkurencí mezi větvemi téže rostliny i konkurencí mezi rostlinami. Uvedeme metody pro generování těch orgánů rostlin, které nelze modelovat pouze úsečkami (jako jsou listy a květy a rozšíříme množinu řídících symbolů pro želvičku o příkazy umožňující řídit barvu kreslených útvarů, generovat mnohoúhelníky, vkládat do modelu rostliny předem definované tvary atd.

Řídící mechanismy v rostlinách

Předpokládejme, že chceme vytvořit L-systém modelující jednoduchou rostlinu s nerozvětveným stonkem, ze kterého přímo vyrůstají listy, a s jediným květem na vrcholu stonku. Viz obr. (2.1). Tento model můžeme popsat následujícím L-systémem:

$$\begin{aligned} &\omega : a \\ &p_1 : a \rightarrow I[L]a \\ &p_2 : a \rightarrow I[L]A \\ &p_3 : A \rightarrow K \end{aligned} \qquad \textbf{(2.1)}$$

Symbol *a* reprezentuje růstový vrchol. Během vývoje rostliny produkuje tento růstový vrchol v každém derivačním kroku jeden nový segment stonku reprezentovaný symbolem *I* a jeden list reprezentovaný symbolem *L* (pravidlo p_1). V nějakém okamžiku vývoje se růstový vrchol *a* přemění na zárodek květu reprezentovaný symbolem *A* (pravidlo p_2) a ten se v následujícím derivačním kroku přepíše na symbol *K* reprezentující květ (pravidlo p_3). Okamžik, ve kterém bude ve vývoji rostliny použito pravidlo p_2 a růstový vrchol se změní na zárodek květu, není výše uvedeným systémem specifikován. V následujících částech uvedeme některé mechanismy umožňující určit okamžik podobných vývojových změn.

V biologii se tyto mechanismy rozdělují do dvou tříd v závislosti na způsobu přenášení informací, podle kterých se okamžik vývojové změny určuje. První způsob se nazývá *buněčná dědičnost* a označuje předávání informací mezi buňkami od předků k jejich následníkům. Naproti tomu *interakcí* nazýváme mechanismus výměny informací mezi sousedními buňkami (např. prostřednictvím živin nebo hormonů). Zatímco buněčnou dědičnost lze modelovat pomocí 0L-systémů, k modelování interakcí je nutné použít kontextové L-systémy. Nyní popíšeme zabudování těchto mechanismů do L-systému (2.1).

Obr. 2.1 Schéma jednoduché rostliny

Stochastický mechanismus

Prvním způsobem implementace vývojových změn, který zde uvedeme je stochastický mechanismus. K modelování použijeme stochastický L-systém a určíme, že pravděpodobnost změny růstového vrcholu v zárodek květu je v každém derivačním kroku π_1. Pravděpodobnost, že růstový vrchol zůstane ve vegetativní fázi tedy bude $\pi_2 = 1 - \pi_1$. Výsledný stochastický L-systém bude vypadat takto:

$$\begin{aligned} &\omega \; : \; a \\ &p_1 : a \rightarrow (\,\pi_2\,)\; I[L]a \\ &p_2 : a \rightarrow (\,\pi_1\,)\; I[L]A \\ &p_3 : A \rightarrow K \end{aligned} \qquad \textbf{(2.2)}$$

Rozložení pravděpodobnosti (π_1, π_2) je třeba určit experimentálně nebo pozorováním.

Vliv prostředí

U některých rostlin je přechod z vegetativního stadia do stadia květu ovlivněn okolním prostředím, např. teplotou, množstvím vláhy nebo délkou dne. Tohoto efektu můžeme dosáhnout tak, že v každé fázi vývoje použijeme jinou množinu pravidel. Jako příklad opět uvedeme rozšíření L-systému (2.1). Při modelování vegetativní fáze vývoje používáme tabulku 1 při přechodu do stadia květu ji zaměníme za tabulku 2.

$$\omega : a \qquad \textbf{(2.3)}$$

Tabulka 1	Tabulka 2
$p_1 : a \rightarrow I[L]a$	$p_1 : a \rightarrow I[L]A$
	$p_2 : A \rightarrow K$

Poznamenejme, že tento mechanismus neřeší zcela problém určení času vývojové změny, protože mimo L-systém musí existovat další mechanismus, který v každém derivačním kroku určí, která tabulka pravidel se bude používat.

Časový mechanismus

Časový mechanismus je založen na předpokladu, že růstový vrchol se vyvíjí a prochází sérií stavů $a_0 \ldots a_n$. Zatímco stavy $a_0 \ldots a_{n-1}$ jsou vegetativní, ve stavu a_n dochází k přeměně v zárodek květu. Odpovídající L-systém má tento tvar:

$$\begin{aligned} &\omega : a_0 \\ &p_1 : a_i \rightarrow I[L]a_{i+1} \quad 0 \leq i < n \\ &p_2 : a_n \rightarrow I[L]A \\ &p_3 : A \rightarrow K \end{aligned} \qquad \textbf{(2.4)}$$

Podle tohoto modelu růstový vrchol „počítá“ kolik segmentů vyprodukoval a při dosažení určité hodnoty se přemění v květ. V přírodě takovéto rostliny existují. Před začátkem vývoje květu vyprodukují konstantní počet listů.

Mechanismus hromadění látek v růstovém vrcholu

Tento mechanismus je založen na předpokladu, že v růstovém vrcholu se během vývoje hromadí látka nebo buňky určitého typu, které při určité koncentraci iniciují vývoj květu. Ačkoli motivace je jiná, lze tento mechanismus modelovat L-systémem (2.4). Zde uvedeme jinou možnost, a to modelováni tohoto mechanismu pomocí parametrického L-systému.

$$\begin{aligned} &\omega : a(0) \\ &p_1 : a(c) : c < C \rightarrow I\,[L]\,a\,(c+d) \\ &p_2 : a(c) : c \geq C \rightarrow I[L]A \\ &p_3 : A \;\; : \;\; 1 \qquad \rightarrow K \end{aligned} \qquad \textbf{(2.5)}$$

Parametr c reprezentuje koncentraci příslušné látky ve vrcholu, konstanta d přírůstek koncentrace v jednom derivačním kroku a konstanta C koncentraci, při které dojde k přeměně růstového vrcholu na zárodek květu.

Vývoj řízený signály

U některých rostlin je vývojová změna z vegetativní fáze na fázi květu řízena signálem, vyslaným z dolní části rostliny k vrcholu. Čas vyslání tohoto signálu můžeme určit některou z předchozích metod, například počítáním derivačních kroků. Příkladem použití tohoto mechanismu je L-systém (2.6).

Růstový vrchol *a* produkuje jeden segment *I* a list *L* za *m* derivačních kroků (pravidla p_1 a p_2). Po *d* derivačních krocích je z kořene rostliny vyslán signál reprezentovaný symbolem *S* (pravidla p_3 a p_4). Signál *S* je přenášen po hlavní ose směrem k vrcholu rychlostí jeden segment za *u* derivačních kroků (pravidla p_5, p_6 a p_7). Když signál dojde k růstovému vrcholu způsobí jeho přeměnu na zárodek květu (pravidlo p_8) a následnou přeměnu v květ (pravidlo p_9). Ačkoli se tento model zdá zbytečně složitý, jsou signály nepostradatelné při modelování vývojových sekvencí složitějších rostlin.

$$
\begin{array}{lll}
\omega : & D(0)\ a(1) & \\
p_1 : & a(i) : i < m & \rightarrow a(i+1) \\
p_2 : & a(i) : i = m & \rightarrow I\ [L]\ a(1) \\
p_3 : & D(i) : i < d & \rightarrow D(i+1) \\
p_4 : & D(i) : i = n & \rightarrow S(1) \\
p_5 : & S(i) : i < u & \rightarrow S(i+1) \\
p_6 : & S(i) : i = u & \rightarrow \varepsilon \\
p_7 : & S(i) < I : i = u & \rightarrow I\ S(1) \\
p_8 : & S(i) < a(j) & \rightarrow I\ [L]\ A \\
p_9 : & A & \rightarrow K
\end{array}
\qquad \textbf{(2.6)}
$$

Modelování rostlinných orgánů

Příkazy pro želvičku, které jsme až dosud uvedli, umožňují generovat pouze obrazce skládající se z úseček. V této kapitole se budeme zabývat modelováním rostlinných orgánů pomocí ploch nebo rovinných polygonů a rozšíříme množinu příkazů želvičky o takové, které umožní tyto tvary generovat.

Vkládání předem definovaných ploch

Tato metoda umožňuje jednoduchým způsobem vkládat do obrazce různé typy ploch nezávisle na způsobu jejich reprezentace. Každou vkládanou plochu, budeme reprezentovat jedním symbolem abecedy systému. Navíc rozšíříme množinu příkazů pro želvičku o symbol '~'. Narazí-li želvička při interpretaci řetězce na tento symbol, vloží do obrazce předdefinovanou plochu reprezentovanou symbolem, který v řetězci následuje. Pozice a orientace vkládané plochy *S* se určí pomocí *kontaktního bodu* P_S a dvou kolmých vektorů

$\mathbf{H_S}$ a $\mathbf{U_S}$, které musí být určeny pro každou vkládanou plochu. Plocha se posune a otočí tak, aby platilo:

$$P_S = P_T \qquad \textbf{(2.7)}$$
$$\mathbf{H_S} = \mathbf{H}$$
$$\mathbf{U_S} = \mathbf{U}$$

kde P_T je pozice želvičky, a vektory **H** a **U** jsou dva ze tří vektorů určujících orientaci želvičky.

Výhodou této metody je možnost vytvoření velmi věrných modelů rostlinných orgánů. Nevýhodou je však jistý nesoulad s filozofií L-systémů. Ten se projevuje např. v tom, že se tyto plochy nemohou vyvíjet stejně jako jiné části modelu. Jestliže potřebujeme modelovat vývoj takto generovaných částí rostliny, musíme pro každý vývojový stupeň vytvořit zvláštní plochu a každý krok vývoje modelovat vložením příslušné plochy.

Vyvíjející se modely

Jak již plyne z názvu, popíšeme v této části modely rostlinných orgánů, které se mohou vyvíjet stejným způsobem jako ostatní části systému.

Pro tento účel zavedeme další nové příkazy pro želvičku.

{ Iniciuje mód vytváření polygonu. Po každém kroku vyvolaném příkazem ‘F’ nebo ‘f’, bude pozice želvičky zaznamenána jako další vrchol právě vytvářeného polygonu.

} Ukončí mód vytváření polygonu. Zařadí do obrazce polygon určený zaznamenanými vrcholy.

G Má stejný význam jako příkaz ‘F’, pozice želvičky však není zaznamenána jako vrchol polygonu.

g Má stejný význam jako příkaz ‘f’, pozice želvičky však není zaznamenána jako vrchol polygonu.

. Zaznamená aktuální pozici želvičky jako vrchol polygonu.

Pro vytváření polygonů se používají dvě základní metody. První z nich je pomocí příkazů ‘F’ a ‘f’ obejít s želvičkou po obvodu polygonu a tím zaznamenat všechny jeho vrcholy. Druhá možnost je pohybovat se s želvičkou pomocí příkazů ‘G’ a ‘g’ po jakési kostře polygonu, skládající se např. z některých jeho úhlopříček a jeho vrcholy zaznamenat pomocí příkazu ‘.’. Příkladem využití první metody popsané v tomto odstavci může být následující L-systém modelující vývoj jednoduchého listu.

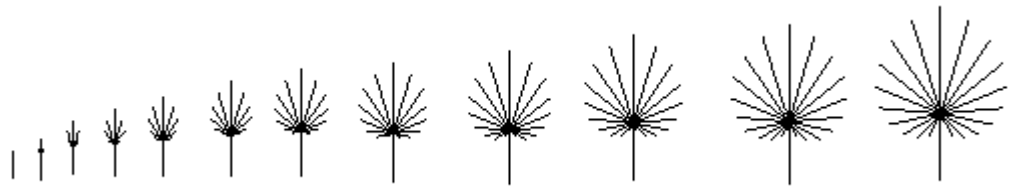
$$\omega : L \qquad \textbf{(2.8)}$$
$$p_1 : L \rightarrow \{ - F X + X - F X - | - F X + X + F X \}$$
$$p_2 : X \rightarrow F X$$

Pravidlo p_1 vytváří list jako vyplněný polygon. Pravidlo p_2 pak zajišťuje lineární růst tohoto listu. Ani tento model však není dostatečně variabilní a s jeho pomocí lze modelovat pouze malé ploché listy. Chceme-li generovat plochu se složitějším okrajem nebo plochu, která není rovinná, je výhodnější vygenerovat stromovou strukturu, která bude tvořit kostru plochy a vrcholy polygonu definovat pomocí příkazu '**.**'. Tímto způsobem je možné pomocí množiny polygonů generovat i poměrně složité plochy.

Pro zjednodušení L-systémů generujících mimi jiné i polygony, modifikujeme význam příkazů '{' a '}'. Příkaz '{' uloží právě vytvářený polygon na zvláštní zásobník (jiný než zásobník na nějž se ukládají stavy želvičky) a jako aktuální bude vytvořen nový prázdný polygon. Příkaz '}' uzavře aktuální polygon, zařadí ho do generovaného obrazce a vybere z vrcholu zásobníku polygon, který se stane aktuálním. Tento způsob interpretace příkazů umožňuje vnořování polygonových závorek '{' a '}' v řetězci. Využití této možnosti ilustruje následující L-systém.

$$\omega : [A] \; [B] \qquad \textbf{(2.9)}$$
$$p_1 : A \rightarrow [+ A \{ .] . C . \}$$
$$p_2 : B \rightarrow [- B \{ .] . C . \}$$
$$p_3 : C \rightarrow G C$$

Axiom obsahuje symboly A a B, ze kterých se podle pravidla p_1 a p_2 vyvíjí levá, resp. pravá polovina listu. Tato pravidla vytvářejí v průběhu vývoje žebra, stále více odkloněná od osy listu. Každá dvě sousední žebra určují trojúhelník, který je zaznamenán a vložen do obrazce pomocí příkazů '.', '{' a '}'. Z těchto trojúhelníků se pak skládá celý list. Na obr. (2.1) je zaznamenán vývoj kostry listu. Samotný list se pak skládá z trojúhelníků určených vždy dvěma sousedními žebry listu.



Obr. (2.1) Vývoj kostry listu generovaného L-systémem (2.9).

Modelování složených listů

Doposud jsme se zabývali modelováním rostlinných orgánů pomocí ploch a polygonů. Chceme-li však generovat některé složené, soběpodobné struktury, např. listy kapradí a

podobně, není tato metoda příliš vhodná, vzhledem k velké členitosti těchto ploch. Při generování těchto ploch se opět vrátíme k obrazcům složeným z úseček. Následující L-systém generuje různé tvary listů kapradí v závislosti na hodnotách konstant *D* a *R*.

$$\omega : A(0) \qquad \textbf{(2.10)}$$
$$p_1 : A(d) : d > 0 \rightarrow A(d\text{-}1)$$
$$p_2 : A(d) : d = 0 \rightarrow F(1)[+A(D)][-A(D)]F(1)A(0)$$
$$p_3 : F(a) \rightarrow F(a * R)$$

Na obr. (2.3) je pět listů vygenerovaných tímto L-systémem. Hodnoty použitých konstant a počty derivačních kroků pro jednotlivé listy jsou uvedeny v tabulce (2.1).

Obr.	D	R	Počet derivačních kroků
a	0	2.00	8
b	1	1.50	14
c	2	1.36	17
d	4	1.23	25
e	7	1.17	38

Tabulka (2.2)

Tropismus

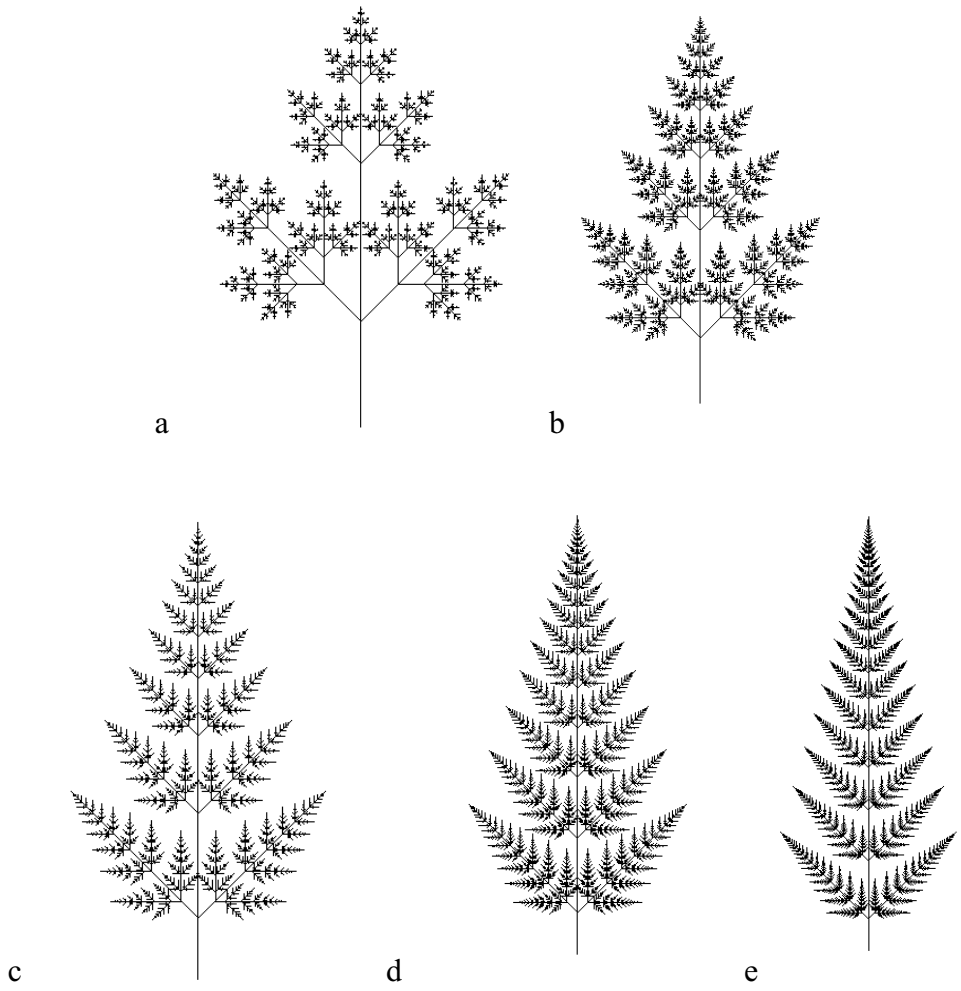
Dosud popsané nástroje pro modelování rostlin popisují tvar rostliny pouze relativně k počátečnímu stavu želvičky. Při vývoji rostlin však obvykle hrají roli i některé absolutní směry. Směr růstu rostliny je ovlivňován gravitací (geotropismus), natočení listů bývá závislé na směru k hlavnímu zdroji světla (fototropismus) atd. Tyto vlivy lze modelovat ve fázi interpretace řetězce následujícím způsobem.

Je dán vektor **T** určující směr tropismu, tj. směr, do kterého je rostlina „přitahována“. Každý segment, který do modelu vkládáme, natočíme směrem k tomuto vektoru o úhel α, jehož velikost lze vyjádřit vztahem $\alpha = |e.\mathbf{S} \times \mathbf{T}|$, kde e je elasticita segmentu, a vektor **S** popisuje směr a délku segmentu.

Další prostředky pro modelování rostlin

Používání L-systémů pro modelování vývoje rostlin vyžaduje oproti generování fraktálních útvarů, geometrických křivek nebo vzorů některá rozšíření metody interpretace řetězců. Několik jich již bylo popsáno v předchozím textu o dalších se stručně zmíníme v této kapitole.

Tvary vzniklé interpretací řetězců generovaných L-systémy jsou pravidelné a přesně odpovídají svému popisu zadanému pomocí pravidel příslušného L-systému. L-systémy však většinou nemohou zachytit všechny vlivy působící na rostlinu a způsobující různé nepravidelnosti v jejím vývoji. Pro dosažení „přírodních tvarů“ vygenerovaných rostlin je vhodné vnést do interpretace řetězce drobné nepřesnosti např. v délkách segmentů nebo úhlech mezi nimi.



Obr. (2.3) Složené listy generované L-systémem (2.10).

Dalším prostředkem k dosažení realistického vzhledu modelů je pokrývání ploch texturami. Vkládání textur může být realizováno obdobným způsobem jako vkládání předdefinovaných ploch.

Specifické pro generování rostlin je opadávání listů a květů. Bylo by velmi složité zajistit, aby se při odpadnutí listu přepsaly všechny symboly podstromu, který tento list reprezentuje, na prázdný řetězec. Jedním z možných řešení je definovat symbol, který označí podstrom jehož je kořenem za neplatný. Jestliže želvička narazí při interpretaci řetězce na tento symbol, celý jeho podstrom ignoruje.

Aplikace lsdraw

Součástí této práce bylo vytvoření programu umožňujícího modelovat vývoj rostlin pomocí L-systémů a podle těchto modelů generovat obrázky rostlin. Výsledkem je aplikace lsdraw, která podporuje používání všech tříd L-systémů popsaných v kapitole 1. V oblasti interpretace řetězce umožňuje používání všech výše popsaných příkazů pro želvičku, randomizaci interpretace, modelování tropismů, nanášení textur, animaci vývojových sekvencí atd. Podrobný popis této aplikace obsahuje následující kapitola.

Kapitola 3

Uživatelský popis aplikace lsdraw

Jak bylo předesláno, aplikace lsdraw poskytuje uživateli prostředky především pro modelování a zobrazování rostlin. Je však možné použít ji i pro generování geometrických obrazců, fraktálních útvarů a křivek a pod. Aplikace vytváří pomocí prostředků knihovny Iris Inventor metodou interpretace řetězce pomocí 3D želvičky z vygenerovaných řetězců třírozměrné objekty. Tyto objekty umožňuje několika způsoby zobrazovat, aplikovat na ně afinní transformace, animovat a ukládat do souborů opět ve formátu Iris Inventor. Knihovna Iris Inventor využívá služeb knihovny Open GL, která využívá grafický hardware počítače Silicon Graphics, na kterém je aplikace implementována.

Aplikace lsdraw se skládá ze dvou samostatných programů *rewrite* a *draw* a programu *lsdraw*, který vytváří grafické uživatelské rozhraní aplikace.

Program rewrite čte ze vstupního souboru L-systém a podle zadaných parametrů vygeneruje pomocí tohoto L-systému řetězec, který uloží v příslušné formě do zadaného výstupního souboru.

Vstupem programu draw jsou data vygenerovaná programem rewrite. Program interpretuje vstupní řetězec pomocí želvičky a generuje 3D objekt ve formátu Iris Inventor. Pokud je ve vstupním souboru uloženo více řetězců, jsou považovány za vývojovou sekvenci. Program vygeneruje jeden objekt pro každý řetězec a umožní animaci této vývojové řady. Program lsdraw tvoří uživatelské rozhraní aplikace. Umožňuje uživateli určit parametry pro oba programy a ty potom spustit tak, že výstup z programu rewrite je přesměrován do roury, ze které pak čte program draw.

Program lsdraw umožňuje uživateli nastavit libovolně všechny parametry, kromě těch, které nastavuje sám pro toto speciální použití obou programů a těch, které jsou při tomto způsobu použití ignorovány.

Program rewrite

Tato část obsahuje úplný uživatelský popis programu. Jsou zde popsány formáty vstupních i výstupních souborů, parametry programu a jejich použití.

Program rewrite umí zpracovávat neparametrické a parametrické kontextové stochastické L-systémy. Podporuje také rozdělování pravidel do tabulek a používání různých tabulek v různých derivačních krocích. Pokud je zpracovávaný L-systém neparametrický, lze to programu oznámit použitím parametru **-s**. Spustíme-li program rewrite s tímto parametrem,

nebudou vytvořeny struktury pro záznam parametrů a podstatně se sníží nároky programu na operační paměť.

Formát vstupního souboru

Základní konstrukty

Při popisování formátu vstupního souboru budeme používat tyto základní konstrukty:

- *identifikátor* - posloupnost písmen a číslic začínající písmenem, dlouhá maximálně 20 znaků.

- *výraz* - číselný výraz vytvořený z čísel, platných identifikátorů konstant, formálních parametrů a závorek zkombinovaných matematickými operátory, jak je popsáno dále v části „Výrazy“.

- *konstantní_výraz* - výraz, který neobsahuje identifikátory formálních parametrů.

- *stprod* - symbol začátku pravidla. Tímto symbolem je znak #, může však být uživatelem předefinován. Symboly, které lze předefinovat budeme nazývat ud-symboly. Další podrobnosti o ud-symbolech jsou uvedeny dále v části „Předefinovatelné lexikální elementy“.

- *lctx* - symbol oddělující levý kontext pravidla. Je to ud-symbol. Standardně je reprezentován znakem <.

- *rctx* - symbol oddělující pravý kontext pravidla. Je to ud-symbol. Standardně je reprezentován znakem >.

- *cond* - symbol oddělující podmínkový výraz. Je to ud-symbol. Standardně je reprezentován znakem : .

- *lpar* - symbol levé závorky. Je to ud-symbol. Standardně je reprezentován znakem (. Jeho předefinování však nemá vliv na zápis výrazů, kde se vždy používá symbol ‘(‘. Symbol *lpar* se používá jako levý omezovač seznamu parametrů a ještě v několika dalších případech, které budou popsány dále, nikdy však jako závorka ve výrazech.

- *rpar* - symbol pravé závorky. Je to ud-symbol. . Standardně je reprezentován znakem). Jeho předefinování však nemá vliv na zápis výrazů, kde se vždy používá symbol ‘)‘. Má obdobné použití jako *lpar*, avšak jako pravý omezovač.

- *arrow* - symbol oddělující levou a pravou stranu pravidla. Je to ud-symbol. Standardně je reprezentován dvojznakem ->.

- *znak_abecedy* - písmeno abecedy L-systému. Může to být identifikátor nebo jeden znak, který není písmenem, číslicí ani ud-symbolem.

- *symbol* - symbol řetězce. Podrobný popis tohoto konstruktu je uveden v kapitole „Symboly řetězce“.

- *formální_symbol_ls* - Symbol ve tvaru, v jakém se vyskytuje na levé straně přepisovacího pravidla. Podrobný popis tohoto konstruktu je uveden v kapitole „Symboly řetězce“.

- *formální_symbol_ps* - Symbol ve tvaru, v jakém se vyskytuje na pravé straně přepisovacího pravidla. Podrobný popis tohoto konstruktu je uveden v kapitole „Symboly řetězce“.

Při popisu formátu vstupního souboru budeme tyto základní konstrukty zapisovat kurzívou. Při zápisu L-systému je třeba je nahradit skutečnými hodnotami. V následujících kapitolách si některé základní konstrukty podrobněji popíšeme.

Používané konvence

Při popisu složených konstruktů pomocí konstruktů základních budeme používat tyto konvence:

[*konstrukt*]	Konstrukt uzavřený v hranatých závorkách je nepovinný.
[*konstrukt*]*	Konstrukt uzavřený v hranatých závorkách s indexem * je nepovinný a může se vícekrát opakovat.
[*konstrukt*]$^{+}$	Konstrukt uzavřený v hranatých závorkách s indexem $^{+}$ je povinný a může se vícekrát opakovat.
axiom	Klíčová slova jsou vytištěna **tučně**.
konstrukt1 | *konstrukt2*	Musí být použit jeden ze dvou uvedených konstruktů.

Výrazy

Výrazy mohou obsahovat čísla, konstanty a formální parametry zkombinované standardním způsobem pomocí závorek, matematických operátorů a některých funkcí. Ve výrazech se rozlišují operandy typu int a float. Typ konstanty je určen typem výsledku konstantního výrazu v definici této konstanty. Formální parametr je vždy typu float. V následující

tabulce jsou uvedeny operátory a funkce, které mohou být použity při vytváření výrazů. Jsou seřazeny podle priority od nejvyšší priority k nejnižší. U všech je uveden typ výsledku. Pokud v tabulce není uvedeno jinak, mohou být operátory aplikovány na libovolnou kombinaci operandů typu int a float.

floor (x) round (x) ceil (x) rnd (x)	Vrací nejvyšší celé číslo menší nebo rovné x. Výsledek je typu int. Zaokrouhlí x a vrátí výsledek této operace. Výsledek je typu int. Vrací nejnižší celé číslo větší nebo rovné x. Výsledek je typu int. Vrací pseudonáhodné číslo typu float z intervalu ⟨ 0, x).
^	Operátor umocňování. Jsou-li oba operandy typu int výsledek je typu int. Jinak je výsledek typu float.
+, -	Unární plus a minus. Výsledek je stejného typu jako operand.
*, /	Operátory násobení a dělení. Jsou-li oba operandy operátoru * typu int je výsledek typu int. Jinak je typu float. Výsledek dělení je vždy typu float.
+, -	Operátory sčítání a odčítání. Jsou-li oba operandy typu int je výsledek typu int. Jinak je typu float.
==, !=, >, <, >=, <=	Relační operátory. Výsledek je vždy typu int.
!	Unární operátor logické negace. Operand musí být typu int. Výsledek je typu int.
&&	Operátor logického součinu. Oba operandy musí být typu int. Výsledek je typu int.
||	Operátor logického součtu. Oba operandy musí být typu int. Výsledek je typu int.

Předefinovatelné symboly (ud-symboly)

Ud-symbol je lexikální element jehož zápis může být uživatelem předefinován. Tyto symboly se vyskytují v takovém kontextu, kdy je nelze rozlišit od symbolů abecedy L-systému se stejným zápisem. Pokud tedy abeceda L-systému takové symboly obsahuje, je nutno zápis příslušného lexikálního elementu předefinovat. Ud-symbol lze předefinovat na:

- znak, který není písmenem ani číslicí kromě znaků reprezentujících matematické operátory, tj. + - * / ^ < > !
- dvojznak neobsahující písmeno ani číslici, kromě dvojznaků reprezentujících matematické operátory, tj. == != >= <= && ||

- identifikátor.

Předefinování ud-symbolů se provádí zadáním příslušných parametrů při volání programu rewrite. Následující tabulka obsahuje default zápisy jmen ud-symbolů a jména parametrů, kterými lze jednotlivé ud-symboly změnit.

ud-symbol	*stprod*	*lctx*	*rctx*	*lpar*	*rpar*	*cond*	*arrow*
zápis	#	<	>	(	)	:	->
parametr	-S	-L	-R	-F	-G	-C	-A

Symboly řetězce

Program rewrite umožňuje pomocí parametru **-s** rozlišit, zda je L-systém na vstupu parametrický či neparametrický a podle toho se mění syntaxe konstruktů *symbol*, *symbol_ls* a *symbol_ps*. Při variantě bez parametrů je syntaxe těchto konstruktů následující:

symbol := znak_abecedy
formální_symbol_ls := znak_abecedy
formální_symbol_ps := znak_abecedy

Při variantě s parametry obsahuje zápis symbolu řetězce navíc seznam parametrů. Symboly s parametry mají tuto syntaxi:

symbol := znak_abecedy [*lpar parametr* [, *parametr*]* *rpar*]
parametr := konstantní_výraz

formální_symbol_ls := znak_abecedy [*lpar form_param_ps* [, *form_ param_ls*]* *rpar*]
form_param_ls := identifikátor

formální_symbol_ps := znak_abecedy [*lpar form_param_ps* [, *form_param_ps*]* *rpar*]
form_param_pxzs := výraz

Pokud má symbol parametry, zapisuje se jejich seznam mezi symboly *lpar* a *rpar*, pokud symbol žádné parametry nemá, neobsahuje jeho zápis ani tyto symboly. Symboly jsou v programu rewrite rozlišovány pouze podle znaku abecedy, nikoli podle počtu parametrů, jak je uvedeno v definici parametrických L-systémů.

Klíčová slova

Klíčová slova nesmějí být použita jako identifikátory v žádné části popisu L-systému. Jako klíčová slova má program rewrite vyhrazeny tyto identifikátory:

axiom	**ceil**	**const**	**default**	**ignore**	**include**
rnd	**round**	**tableoff**	**tableon**	**trunc**	

Struktura vstupního souboru

Popis L-systému obsažený ve vstupním souboru může obsahovat tyto části:

- sekce definic konstant - začíná klíčovým slovem **const**
- sekce symbolů ignorovaných při testování kontextu - začíná klíčovým slovem **ignore**
- sekce symbolů s default parametrem - začíná klíčovým slovem **default**
- definice axiomu - začíná klíčovým slovem **axiom**
- sekce definující příslušnost přepisovacích pravidel k jednotlivým tabulkám - začíná klíčovým slovem **tabbleon** nebo **tableoff**
- definice přepisovacích pravidel

Sekce definice konstant

Tato sekce má tvar:

const [*identifikátor* = *konstantní_výraz;*]*

kde *identifikátor* je identifikátor konstanty. Hodnota a typ konstanty jsou určeny hodnotou a typem výsledku konstantního výrazu. Identifikátor konstanty je platný od místa definice do konce souboru.

Sekce definice symbolů ignorovaných při testování kontextu

Tato sekce má tvar:

ignore [*znak_abecedy*]*

Za klíčovým slovem ignore následuje seznam *znaků_abecedy* systému, které mají být v řetězci při testování kontextu ignorovány. Tato sekce musí ležet před definicemi axiomu a všech přepisovacích pravidel.

Sekce definice symbolů s default parametrem

Symboly s default parametrem jsou symboly, které mají jeden parametr, ale v popisu L-systému se mohou vyskytovat bez uvedení jeho hodnoty. Program v tom případě použije default hodnotu definovanou v tomto úseku. Syntaxe této sekce je následující:

default [*symbol*]*

Možnosti definovat default parametr lze s výhodou využít v rozsáhlejších systémech u příkazů pro želvičku. Jestliže se nějaký symbol vyskytuje v převážné většině případů s jednou hodnotou parametru, můžeme tuto hodnotu určit jako default.

Tato sekce musí ležet před definicí axiomu i všech přepisovacích pravidel. L-systém nesmí sekci **default** obsahovat, pokud byl program rewrite spuštěn s parametrem **-s**.

Axiom

V popisu L-systému se musí vyskytovat právě jedna definice axiomu. Její syntaxe je následující:

axiom [*symbol*]+

Definice tabulek pravidel

Program rewrite umožňuje rozdělit přepisovací pravidla L-systému do několika tabulek a v různých derivačních krocích pak používat různé tabulky. Každé pravidlo může patřit do několika tabulek. Při rozvíjení řetězce může být v jednom derivačním kroku použito také více tabulek. V každém derivačním kroku je použita předdefinovaná tabulka **common**. Použití ostatních tabulek se určuje pomocí parametru **-T**, při volání programu rewrite.

Výskyt identifikátoru tabulky v sekci definice aktivních tabulek, znamená že všechna následující pravidla patří (mimo jiné) do této tabulky. To platí až do místa, kde se identifikátor této tabulky vyskytne v sekci deaktivace tabulek. Sekce definice aktivních tabulek má syntaxi:

tableon *identifikátor* [, *identifikátor*]*

Sekce deaktivace tabulek pravidel má syntaxi:

tableoff *identifikátor* [, *identifikátor*]*

Každý identifikátor tabulky pravidel je definován svým prvním výskytem v jedné z těchto sekcí. Na začátku čtení vstupního souboru je aktivní pouze tabulka **common**, která je pak také používána při rozvíjení řetězce v každém derivačním kroku. To znamená, že pokud uživatel nechce používat tabulky pravidel, nemusí se o ně starat ani v definici L-systému ani při volání programu rewrite.

Definice přepisovacích pravidel

Definice přepisovacího pravidla má syntaxi:

pravidlo :=
stprod [*l_kontext*] *přepisovaný_symbol* [*p_kontext*] [*podmínka*] *pravá_strana*

přepisovaný_symbol :=
formální_symbol_ls

l_kontext :=
[*formální_symbol_ls*]* *lctx*

p_kontext :=
rctx [*formální_symbol_ls*]*

podmínka :=
cond lpar výraz rpar

pravá_strana :=
deterministická_ps | *stochastická_ps*

deterministická_ps :=
arrow [*formální_symbol_ps*]*

stochastická_ps :=
[*arrow lpar pravděpodobnost rpar* [*formální_symbol_ps*]*]$^{+}$

pravděpodobnost :=
konstantní_výraz

Vkládání souborů

Vyskytne-li se ve vstupním souboru konstrukt

include "*soubor*"

je tento konstrukt nahrazen obsahem souboru *soubor*. Vkládaní souborů lze vnořovat tj. vkládaný soubor může opět obsahovat tento konstrukt.

Výstup z programu rewrite

Výstupem z programu rewrite je vygenerovaný řetězec nebo více řetězců tak, jak byly postupně generovány v jednotlivých derivačních krocích. Uživatel může pomocí parametru **-i** určit, po kolika derivačních krocích má být řetězec zapisován na výstup. První řetězec, který bude zapsán do výstupního souboru je potom automaticky určen tak, aby posledním

vypisovaným byl řetězec vygenerovaný v posledním derivačním kroku. Dále může uživatel zvolit, zda budou tyto řetězce zapsány všechny do jednoho souboru nebo každý do zvláštního. Pokud je každý řetězec vypisován do zvláštního souboru, jsou jména výstupních souborů generována ze jména výstupního souboru zadaného parametrem **-o**, připojením čísla označujícího pořadí vypisovaného řetězce ke jménu výstupního souboru. Pokud je jako výstupní soubor zadán standardní výstup, jsou všechny řetězce vypsány na standardní výstup. Řetězec může být zapsán na výstup v binárním nebo textovém formátu.

Výstup v textovém formátu

Je-li nastaven textový výstup, jsou neparametrické symboly řetězce zapisovány do výstupního souboru ve formátu

znak_abecedy

a parametrické symboly ve formátu

znak_abecedy lpar seznam_parametrů rpar

kde *seznam_parametrů* jsou hodnoty parametrů oddělené čárkami. Symboly jsou ve výstupním souboru odděleny mezerou nebo znakem nového řádku. Počet symbolů na jednom řádku lze nastavit pomocí parametru **-l**. Řetězec je zakončen dvěma znaky nového řádku.Výstup v textovém formátu se nastavuje parametrem **-x**.

Výstup v binárním formátu

Při výstupu v binárním tvaru se do souboru zapíše nejprve hlavička a za ní následují vygenerované řetězce. Hlavička má tvar:

Počet bytů	Popis
4 byty	Identifikační číslo binárního výstupního souboru programu rewrite.
2 byty	Počet řetězců v souboru.
2 byty	Kód symbolu **eos** (end of string), který ukončuje každý řetězec.
2 byty	Počet položek v tabulce symbolů.
n * 23 bytů	Tabulka symbolů. Každá položka se skládá z 21 bytového pole znaků, které obsahuje nulou ukončený řetězec se jménem symbolu a 2 bytové číslo, které zaznamenává aritu symbolu.

Za hlavičkou souboru jsou pak uloženy jednotlivé řetězce. Každý řetězec je ukončen symbolem **eos**. Každý symbol je zapsán jako dvoubytový kód symbolu, který je po

vynulování dvou nejvyšších bitů indexem do tabulky symbolů v hlavičce, za kterým následují hodnoty parametrů. Každý parametr je reprezentován osmibytovým číslem typu double float.

Parametry programu

Volání programu rewrite má tvar:

rewrite *vstupní_soubor* [*volby*]

Parametr *vstupní_soubor* je povinný, všechny ostatní volby jsou nepovinné. Pokud je některá volba uvedena při volání vícekrát, je použita poslední uvedená hodnota. Výjimkou z tohoto pravidla jsou volby **-f** a **-t**, jak je patrno z jejich popisu.

Následující tabulka obsahuje popis všech voleb programu rewrite.

Formát	Popis
-o [*výstupní_soubor*]	Jméno výstupního souboru. Pokud je tato volba zadána bez jména souboru nebo zcela chybí, bude jako výstupní soubor použit standardní výstup.
-x	Výstup v textovém formátu. Pokud nebude tato volba zadána, výstup bude v binárním formátu.
-d *počet_kroků*	Počet derivačních kroků. Default hodnota je 10.
-i *počet_kroků*	Počet kroků, po kterém má být řetězec zapsán do výstupního souboru. Pokud není tento parametr zadán, je na výstup zapsán pouze řetězec vygenerovaný posledním derivačním krokem.
-a	Je-li tato volba použita, budou všechny řetězce zapsány do jednoho výstupního souboru. V opačném případě je každý řetězec zapsán do zvláštního souboru.
-l *počet_symbolů*	Počet symbolů na řádku v textovém výstupním souboru.
-s	Je-li zadána tato volba, L-systém na vstupu bude považován za neparametrický.
-t *číslo_kroku* [*identifikátor_tabulky*]*	Od derivačního kroku *číslo_kroku* bude používána tabulka pravidel **common** a tabulky, jejichž identifikátory jsou uvedeny v tomto parametru. Tento parametr může být při volání programu zadán vícekrát a nastavení tabulek tak může být v průběhu derivace několikrát změněno.

-f *param_soubor*	Místo tohoto parametru jsou vloženy parametry, které obsahuje soubor *param_soubor*.
-S *ud_symbol*	Předefinování ud-symbolu *stprod.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-L *ud_symbol*	Předefinování ud-symbolu *lctx.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-R *ud_symbol*	Předefinování ud-symbolu *rctx.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-C *ud_symbol*	Předefinování ud-symbolu *cond.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-A *ud_symbol*	Předefinování ud-symbolu *arrow.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-F *ud_symbol*	Předefinování ud-symbolu *lpar.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-G *ud_symbol*	Předefinování ud-symbolu *rpar.* Řetězec *ud_symbol* je nový zápis tohoto symbolu
-H *znak_abecedy*	Touto volbou lze předefinovat znak abecedy systému, který reprezentuje příkaz uložení stavu želvičky do zásobníku. Default hodnota je [. Program musí tento symbol znát, aby mohl správně určovat kontext symbolů v řetězci.
-P *znak_abecedy*	Touto volbou lze předefinovat znak abecedy systému, který reprezentuje příkaz vybrání stavu želvičky ze zásobníku. Default hodnota je]. Program musí tento symbol znát, aby mohl správně určovat kontext symbolů v řetězci.

Program draw

Program draw provádí interpretaci řetězce pomocí 3D želvičky. Vstupem pro tento program jsou data v binárním formátu vygenerovaná programem rewrite. Program generuje 3D objekt skládající se z úseček a polygonů, ve vnitřním formátu Iris Inventor. Vygenerovaný objekt lze buď zobrazit nebo uložit do souboru v textovém nebo binárním formátu. Pokud vstupní soubor obsahuje více řetězců, je pro každý řetězec vygenerován jeden objekt. Tyto objekty lze pak současně zobrazit nebo zvolit animaci této vývojové sekvence.

Symboly interpretované želvičkou

Množina příkazů pro želvičku obsahuje všechny dosud popsané příkazy a navíc příkazy pro nastavení barvy čar ve formátu RGB, vkládání textur a další příkazy usnadňující generování a zobrazování rostlinných tvarů. Pro každý parametrický symbol je určen počet parametrů, které ovlivňují interpretaci tohoto symbolu. Má-li symbol méně parametrů, jsou použity default hodnoty zadané jako parametry při spuštění programu. Pokud tyto parametry nebyly zadány, použijí se default hodnoty programu.

V následující tabulce jsou uvedeny symboly interpretované želvičkou programu draw. U každého symbolu je uveden počet parametrů, které ovlivňují interpretaci symbolu, default hodnoty parametrů a popis odpovídající akce.

Symbol	Počet parametrů	Default hodnota	Popis akce
F	1	1	Želvička se přesune ve směru vektoru **H** o krok délky, kterou udává první parametr symbolu. Do obrazce vloží úsečku určenou původní a novou pozicí želvičky. Pokud je otevřen mód záznanu polygonu, nová pozice bude zaznamenána jako vrchol polygonu.
f	1	1	Želvička se přesune ve směru vektoru **H** o krok délky, kterou udává první parametr symbolu. Pokud je otevřen mód záznanu polygonu, nová pozice bude zaznamenána jako vrchol polygonu.
G	1	1	Želvička se přesune ve směru vektoru **H** o krok délky, kterou udává první parametr symbolu. Do obrazce vloží úsečku určenou původní a novou pozicí želvičky.
g	1	1	Želvička se přesune ve směru vektoru **H** o krok délky, kterou udává první parametr symbolu.
+	1	45	Želvička se otočí doleva kolem osy **U**, o úhel jehož velikost udává první parametr symbolu.
-	1	45	Želvička se otočí doprava kolem osy **U**, o úhel jehož velikost udává první parametr symbolu.
^	1	45	Želvička se otočí nahoru kolem osy **L**, o úhel jehož velikost udává první parametr symbolu.

&	1	45	Želvička se otočí dolů kolem osy **L**, o úhel jehož velikost udává první parametr symbolu.
/	1	45	Želvička se nakloní doleva kolem osy **H**, o úhel jehož velikost udává první parametr symbolu.
\	1	45	Želvička se nakloní doprava kolem osy **H**, o úhel jehož velikost udává první parametr symbolu.
|	0	-	Želvička se otočí o 180^{o} kolem osy **U**.
$	0	-	Nastaví želvičku do vertikální polohy.
[	0	-	Uloží stav želvičky na zásobník.
]	0	-	Vybere stav želvičky ze zásobníku.
{	0	-	Uloží aktuální polygon na polygonový zásobník a vytvoří nový prázdný polygon, který se stane aktuálním.
}	0	-	Uzavře aktuální polygon a vloží ho do obrazce. Vybere ze zásobníku polygon, který se stane aktuálním.
.	0	-	Zaznamená současnou pozici želvičky jako vrchol aktuálního polygonu.
!	1	-0.1	Změní šířku kreslené čáry o hodnotu prvního parametru symbolu.
setwidth	1	2	Nastaví šířku kreslené čáry na hodnotu prvního parametru symbolu.
’	3	0.1 0.1 0.1	Změní aktuální barvu tak, že přičte hodnoty prvních tří parametrů k jejím RGB složkám.
setcolor	3	0 0.5 0	Nastaví aktuální barvu. Hodnoty prvních tří parametrů tvoří RGB reprezentaci nové aktuální barvy.
@	0	-	Nastaví aktuální texturu. Textura je určena následujícím symbolem a jeho prvním parametrem. Pokud následuje opět symbol ‘@’, bude aktuální textura zrušena. Podrobnosti jsou uvedeny v kapitole „Vkládání předdefinovaných objektů“.

~	0	-	Vloží do obrazce předdefinovaný objekt ve formátu Inventor reprezentovaný následujícím symbolem a jeho prvním parametrem.
%	0	-	„Odřízne“ zbytek větve, tj. ignoruje všechny symboly dokud nenarazí na symbol ‘]’ párový k poslednímu symbolu ‘[‘ před symbolem ‘%’.

Vkládání předdefinovaných objektů a textur

Chceme-li do obrazce vkládat předdefinované obrazce nebo textury je nutné spojit některé symboly se jmény souborů, ve kterých jsou příslušné objekty nebo textury uloženy. Toto propojení se definuje v *asociačních souborech*, jejichž jména jsou pak předána programu draw jako parametry. Obsah asociačního souboru má formát:

obsah_souboru := [*asociační_blok*]*
asociační_blok := start_bloku asociovaný_symbol [*jméno_souboru*]$^{+}$
start_bloku := *****
asociovaný_symbol := znak_abecedy

K jednomu znaku abecedy systému je tedy připojeno několik jmen souborů. Chceme-li vložit do obrazce předdefinovaný objekt nebo texturu, musí řetězec obsahovat příslušný příkaz, následovaný asociovaným symbolem. Jeho parametr je pak indexem do tabulky souborů, připojených k tomuto symbolu.

Vkládání předdefinovaných objektů

Příkaz pro vložení objektu je reprezentován symbolem ‘~’. Popis vkládaného objektu musí být v příslušném souboru uložen v binárním nebo textovém formátu Iris Inventor. Při vkládání objektu se nemění pozice ani orientace želvičky.

Vkládání textur

Příkaz pro vložení textury je reprezentován symbolem ‘@’. Vložená textura je aplikována na celý podstrom tohoto symbolu, pokud není později předefinována nebo zrušena. Zrušení textury se provede příkazem ‘@’ za nímž následuje ještě jednou symbol ‘@’.

Zobrazení vygenerovaných objektů

K zobrazování objektů požívá program draw zobrazovač *Examiner viewer*, který poskytuje knihovna Inventor. Tento zobrazovač umožňuje pomocí myši objekt posouvat, otáčet a měnit jeho velikost. Také lze zvolit zobrazení objektu bez textur, drátový model nebo pouze zobrazení koncových bodů úseček a vrcholů polygonů. Pokud byl programu zadán parametr **-p** tj. animace vývojové sekvence, je na levém okraji okna zobrazovače navíc umístěno pět tlačítek, pomocí kterých lze animaci řídit. Jejich význam je popsán v následující tabulce.

Tlačítko	Popis funkce
>	Zobrazí následující objekt v sekvenci. Po dvojkliku zobrazí poslední objekt v sekvenci.
<	Zobrazí předcházející objekt v sekvenci. Po dvojkliku zobrazí první objekt v sekvenci.
A	Spouští a zastavuje animaci. Pokud animace probíhá, svítí toto tlačítko zeleně.
+	Zvyšuje rychlost animace.
-	Snižuje rychlost animace.

Potlačení přímých linií

Na modelech rostlin mohou působit nepřirozeně dlouhé přímé linie bez jediné nepravidelnosti nebo záhybu. Program draw umožňuje při interpretaci řetězce tyto linie náhodně zohýbat. Jestliže je potlačování přímých linií zapnuto, je při generování souvislé lomené čáry vždy po určitém počtu kroků změněna orientace želvičky o úhel náhodné velikosti. Maximální velikost tohoto úhlu je určena parametrem **-g** a počet segmentů, po kterém je měněna orientace želvičky je určen parametrem **-n**. Zadáním alespoň jednoho z těchto parametrů se zároveň zapíná potlačování přímých linií.

Ohýbání přímých linií není aplikováno na segmenty generované příkazem G. Tento příkaz je určen především pro generování polygonů, při kterém by tyto úpravy mohly způsobit nevhodné efekty, např. překrývání nebo nesouvislost polygonů, které mají tvořit jednu plochu.

Modelování tropismu

Vektor tropismu se zadává parametrem -**r**, za kterém následují tři parametry reprezentující složky vektoru. Délka vektoru udává elasticitu větví, jak byla popsána v části „Tropismus“

v kapitole 2. Při generování obrazce je pak před provedením každého příkazu F nebo G změněna orientace želvičky tak, aby se vektor **H** přiklonil k vektoru tropismu o úhel $\alpha = | \mathbf{H} \times \mathbf{T} |$, kde **T** je vektor tropismu.

Parametry programu draw

Program draw lze spouštět s těmito parametry:

Formát	Popis
vstupní_soubor	Jméno vstupního souboru musí být zadáno jako první parametr. Pokud chybí, program čte ze standardního vstupu.
-o [*výst_soubor*]	Požadavek na zápis objektu do souboru *výst_soubor* ve formátu Inventor. Pokud jméno souboru chybí, data jsou zapsána na standardní výstup.
-t	Výstup v textovém formátu.
-b	Výstup v binárním formátu. Default hodnota.
-v	Požadavek na zobrazení objektu zobrazovačem Examiner viewer. Objekt je zobrazen také v případě, že není zadán žádný z parametrů **-v** a **-o**.
-T *tex_soubor*	Asociační soubor pro textury.
-I *inc_soubor*	Asociační soubor pro vkládané objekty.
-s *délka_kroku*	Délka kroku želvičky pro příkazy F, G, f, g pokud se v řetězci nacházejí bez parametrů.
-a *úhel*	Úhel o který se želvička otočí při vykonávání příkazů +, -, ^, &, \, / pokud se v řetězci nacházejí bez parametrů.
-W *šířka_čáry*	Hodnota na, kterou bude nastavena šířka čáry příkazem *sewidth* bez parametru.
-w *šířka_čáry*	Hodnota, která bude přičtena k šířce čáry příkazem ! bez parametru.
-C *r g b*	Barva, která bude nastavena jako aktuální příkazem *setcolor* s méně než 3 parametry.

-c *r g b*	Hodnoty, které budou přičteny k jednotlivým složkám aktuální barvy příkazem ’ s méně než 3 parametry.
-r *x y z*	Vektor tropismu.
-m *x y z*	Vektor, o který bude posunut každý další objekt při současném zobrazení všech objektů.
-p	Požadavek na animaci vývojové sekvence.
-n *počet_seg*	Počet segmentů generovaných příkazem F, po kterém má být provedeno náhodné pootočení želvičky při eliminaci přímých linií.
-g *max_úhel*	Maximální úhel, o který může být želvička otočena při eliminaci přímých linií.
-f *prm_soubor*	Místo tohoto parametru jsou vloženy parametry, které obsahuje soubor *prm_soubor*.

Program lsdraw

Program lsdraw tvoří uživatelské rozhraní aplikace. Umožňuje uživateli zadat parametry pro programy rewrite a draw a spustit je tak, že program rewrite zapisuje do roury, ze které program draw čte.

Uživatelské rozhraní

Po spuštění aplikace lsdraw se zobrazí hlavní okno aplikace, které obsahuje vstupní políčka, tlačítka, přepínací tlačítka a jedno seznamové okénko, jejichž prostřednictvím se zadávají základní parametry, tj. vstupní a výstupní soubory, výstupní formáty atd. Vedle každého políčka pro vstup jména souboru je tlačítko Browse, které otevírá dialogové okno pro výběr souboru. V dolní části hlavního okna jsou umístěna čtyři tlačítka s globálním významem. Jejich funkce je popsána v následující tabulce.

Tlačítko	Popis funkce
Start	Spustí kolonu rewrite | draw s právě nastavenými parametry.
Save prm	Otevře okno pro výběr souboru a uloží všechny parametry do vybraného souboru.
Load prm	Otevře okno pro výběr souboru a přečte všechny parametry z vybraného souboru.
Quit	Ukončí aplikaci.

Zadávání parametrů pro program rewrite

Část hlavního okna určená pro zadávání parametrů pro program rewrite je nadepsána REWRITE PARAMETERS a obsahuje tyto prvky:

Popis	Typ	Význam (parametr)
Input file	Vstupní políčko	Jméno vstupního souboru
Step count	Vstupní políčko	Počet derivačních kroků (**-d**)
Output step	Vstupní políčko	Počet kroků, po kterém je prováděn výstup řetězce (**-i**)
Non-parametric symbols	Přepínací tlačítko	Nastavení neparametrických symbolů (**-s)**
Tables	Seznamové okénko	Obsahuje seznam změn aktivních tabulek pravidel ve formátu *derivační_krok* [*identifikátor_tabulky*]* (**-t**)
Add	Tlačítko	Tlačítko pro přidávání položek do seznamového okénka Tables
Delete	Delete	Stiskem tohoto tlačítka lze vymazat všechny označené položky v seznamovém okénku Tables

Tlačítkem Lexical lze otevřít dialogové okno pro předefinování ud-symbolů. Toto okno obsahuje prvky:

Popis	Typ	Význam (parametr)
Symbol *stprod*	Vstupní políčko	Změna ud-symbolu *stprod* (**-S)**
Symbol *lctx*	Vstupní políčko	Změna ud-symbolu *lctx* (**-L)**
Symbol *rctx*	Vstupní políčko	Změna ud-symbolu *rctx* (**-R)**
Symbol *cond*	Vstupní políčko	Změna ud-symbolu *cond* (**-C)**
Symbol *arrow*	Vstupní políčko	Změna ud-symbolu *arrow* (**-A)**
Symbol *lpar*	Vstupní políčko	Změna ud-symbolu *lpar* (**-F)**
Symbol *rpar*	Vstupní políčko	Změna ud-symbolu *rpar* (**-G)**

Pokud zůstane některé vstupní políčko nevyplněné nebo je přepínací tlačítko vypnuto, nebude odpovídající parametr při volání programu zadán. Nevyplnění políčka Input file způsobí chybové hlášení a kolona nebude spuštěna.

Program rewrite je vždy spouštěn s parametry **-o** (zápis na standardní výstup) a **-a** (zápis všech řetězců do jednoho výstupního souboru). Ostatní parametry nejsou při volání z lsdraw používány proto, že jejich hodnota by byla v daném kontextu ignorována (např. parametr **-l**) nebo proto, že program rewrite je nutno pro použití v aplikaci lsdraw spustit s příslušnými default hodnotami (to platí např. pro parametry **-x**, **-b**, **-H**, **-P**).

Zadávání parametrů pro program draw

Část hlavního okna určená pro zadávání parametrů pro program draw je nadepsána DRAW PARAMETERS a obsahuje tyto prvky:

Popis	Typ	Význam (parametr)
Output file	Vstupní políčko	Jméno výstupního souboru. Bude použito pouze pokud je zapnuto přepínací tlačítko Output to file.

Includes file	Vstupní políčko	Asociační soubor pro vkládané objekty (-**I**)
Textures file	Vstupní políčko	Asociační soubor pro textury (-**T**)
View	Přepínací tlačítko	Zobrazení vygenerovaných objektů. (-**v**)
Output to file	Přepínací tlačítko	Zápis vygenerovaných objektů do souboru. (-**o**)
Text output	Přepínací tlačítko	Výstup v textovém formátu. (-**t**)
Binary output	Přepínací tlačítko	Výstup v binárním formátu (-**b**)

Stiskem tlačítka More lze otevřít okno pro zadávání dalších parametrů programu draw. Toto okno popisuje následující tabulka.

Popis	Typ	Význam (parametr)
Step len	Vstupní políčko	Default hodnota délky kroku želvičky. (-**s**)
Angle	Vstupní políčko	Default hodnota úhlu otočení. (-**a**)
Set width	Vstupní políčko	Default hodnota parametru příkazu setwidth. (-**W**)
Change width	Vstupní políčko	Default hodnota parametru příkazu '!' (-**w**)
Set color	Vstupní políčko	Default hodnoty parametrů příkazu setcolor ve tvaru *r g b* (-**C**)
Change color	Vstupní políčko	Default hodnoty parametrů příkazu ' ve tvaru *r g b* (-**c**)
Move	Vstupní políčko	Vektor posunutí dalšího objektu při současném zobrazování všech objektů. (-**m**)
Tropism	Vstupní políčko	Vektor tropismu. (-**r**)
Rnd period	Vstupní políčko	Počet segmentů, po kterém je želvička náhodně pootočena při potlačování přímých linií. (-**n**)
Max rnd angle	Vstupní políčko	Maximální úhel, o který může být želvička pootočena při potlačování přímých linií. (-**g**)
Animation	Přepínací tlačítko	Požadavek na animaci vývojové sekvence. Pokud je toto tlačítko vypnuté, všechny objekty budou zobrazeny současně. (-**p**)

Parametry programu lsdraw

Parametry programu lsdraw určují adresář, ve kterém se nacházejí programy rewrite a draw. Pokud některý z těchto parametrů chybí, předpokládá se, že se příslušný program nachází v aktuálním adresáři.

Popis	Formát
-r *adresář*	Jméno adresáře, ve kterém se nachází program rewrite.
-d *adresář*	Jméno adresáře, ve kterém se nachází program draw.

Kapitola 4

Dokumentace k aplikaci lsdraw

Tato kapitola poskytuje stručný přehled o algoritmech a datových strukturách používaných v aplikaci. Podrobnější popisy jednotlivých funkcí lze nalézt v komentářích ve zdrojových textech.

Sdílené moduly

Tato část obsahuje popis několika modulů, které jsou použity ve více programech. Jsou to moduly args, darray, errstr, generr, hash a svalloc.

Modul args

Poskytuje funkce pro zpracování parametrů programu. Podporuje jednoznakové volby následované dalšími parametry, tj. formát

$$\textit{-znak}\ [\ \textit{prm}\ ]^{*}$$

který je použit pro zadávání parametrů u všech programů aplikace.

Modul darray

Tento modul implementuje dynamicky alokované pole libovolných prvků, jehož velikost se v případě potřeby automaticky mění.

Modul errstr

Obsahuje řetězce chybových hlášení pro všechny programy.

Modul generr

Poskytuje funkce *Error* a *Fatal*. Funkce *Error* vypíše na standardní chybový výstup jméno programu a řetězec, který dostane jako parametr. Funkce *Fatal* navíc ukončí program.

Modul hash

Poskytuje funkce pro hašování řetězců.

Modul svalloc

Poskytuje bezpečné funkce pro alokace paměti. Jsou to funkce *savemalloc*, *savecalloc*, *saverealloc* a *savefree*. Funkce pro alokaci paměti volají standardní funkce *malloc*, *calloc* a *realloc* a při nedostatku paměti ukončí program voláním funkce *Fatal*. Funkce *savefree* kontroluje zda není uvolňován NULL pointer.

Program rewrite

Program rewrite pracuje ve třech fázích. V první fázi načítá vstupní parametry a uloží jejich hodnoty do globálních proměnných, podle jejichž hodnot se pak řídí další práce programu. Ve druhé fázi načítá vstupní soubor a vytváří vnitřní reprezentaci L-systému. Ve třetí fázi pak provádí rozvíjení řetězce a zapisuje výsledky na výstup.

Fáze zpracování argumentů

Pro zpracování parametrů programu se využívá modul rarg.

Modul rarg

Modul rarg poskytuje několik globálních proměnných, jejichž hodnoty ovlivňují práci programu v dalších dvou fázích. Dále poskytuje funkci *ProcArgs*, která přečte parametry programu a podle nich nastaví hodnoty těchto proměnných.

Fáze zpracování vstupního souboru

Tuto fázi výpočtu zajišťují moduly lexical, filestk, parser, tables, product, genexpr, evaluate, tempvar a error.

Modul lexical

Modul lexical zajišťuje lexikální analýzu vstupního souboru. Pro tento účel poskytuje funkci *GetLexItem* a globální proměnnou *LexItem.* Voláním funkce *GetLexItem* se načte další lexikální element a jeho reprezentace se uloží do proměnné *LexItem*.

Pro účely výpisu chybových hlášení uchovává modul údaje o pozici posledního a předposledního lexikálního elementu. Poskytuje funkci *PrintErrLine*, která podle parametrů vypíše do příslušného souboru řádek obsahující poslední nebo předposlední lexikální element.

Modul lexical dále zajišťuje zpracování konstruktu **include**, tj. vložení obsahu jiného souboru na místo tohoto konstruktu.

Modul filestk

Tento modul poskytuje zásobník pro uchování jmen souborů při vnořování konstruktu **include**.

Modul parser

Obsahuje funkce pro načtení a zpracování všech sekcí vstupního souboru. Poskytuje funkci *ReadFile*, která přečte celý vstupní soubor a vytvoří jeho vnitřní reprezentaci.

Modul tables

Obsahuje struktury pro uchovávání a práci s identifikátory a znaky abecedy systému. V programu se vyskytují identifikátory znaků abecedy, formálních proměnných, celočíselných konstant, konstant typu float a tabulek pravidel.

Počet tabulek pravidel je omezen počtem bitů proměnné typu long int. Vzhledem k jejich omezenému počtu jsou identifikátory tabulek pravidel ukládány v poli pevné délky a nejsou vytvořeny žádné struktury pro urychlování práce s těmito identifikátory.

Základní informace o identifikátorech konstant a znacích abecedy jsou uloženy v hašovací tabulce *idtab*. Do této tabulky se ukládají struktury obsahující jméno, typ a kód identifikátoru. Kód identifikátoru je pak indexem do tabulky rozšiřujících informací dané typem identifikátoru.

Rozšiřující informace pro znaky abecedy jsou uloženy v dynamickém poli *ltab*. Indexem do této tabulky je kód znaku abecedy s vynulovanými nejvyššími dvěma bity. V těchto bitech se ukládá informace o tom zda se znak abecedy vyskytl v sekcích **ignore** nebo **default**. Záznam v tabulce *ltab* obsahuje tyto informace: identifikátor znaku abecedy, jeho aritu a ukazatele na seznamy kontextových a bezkontextových pravidel v nichž se tento znak vyskytuje jako přepisovaný symbol.

Rozšiřující informace o identifikátorech konstant jsou uloženy v dynamických polích *ic* pro celočíselné konstanty a *fc* pro konstanty typu float. Záznamy v těchto tabulkách obsahují pouze hodnotu příslušné konstanty.

Identifikátory formálních proměnných se ukládají do hašovací tabulky *vartab*. Záznamy v této tabulce obsahují identifikátor a kód proměnné. Kód se proměnným přiřazuje podle pořadí prvního výskytu v příslušném pravidle. Identifikátory proměnných nemají žádné rozšiřující informace. Protože identifikátory proměnných jsou lokální v přepisovacím

pravidle, je nutné tabulku *vartab* po načtení celého pravidla vymazat. To je také důvod proč nejsou identifikátory proměnných ukládány do tabulky *idtab*.

Modul dále obsahuje hašovací tabulky *ignortab* a *deftab* do nichž se ukládají znaky abecedy, které se vyskytly v sekcích **ignore** resp. **default**. Záznamy v tabulce *ignortab* obsahují pouze příslušný identifikátor, záznamy v tabulce *deftab* obsahují navíc default hodnotu parametru pro příslušný symbol. Modul poskytuje řadu funkcí pro vkládání, vyhledávání a další operace s těmito tabulkami.

Modul product

Poskytuje struktury pro reprezentaci přepisovacích paravidel.

Modul genexpr

Poskytuje funkci *CreateExpr*, která čte výraz ze vstupního souboru a vytváří jeho interní reprezentaci pomocí čtveřic. K tomu využívá mimo jiné služeb modulu tempvar, který spravuje pomocné proměnné.

Modul evaluate

Poskytuje funkce pro vyhodnocování výrazů v interním formátu.

Modul error

Poskytuje funkci *GrmError* pro výpis hlášení o chybách ve vstupním souboru. Chyby se dělí do tří skupin : varování, chyba a fatální chyba. Varování neovlivní úspěšné dokončení parsingu. Chyba znemožňuje vytvoření některých struktur a proto nedovolí po ukončení parsingu spuštění derivační fáze programu. Fatální chyba okamžitě ukončí běh programu.

Fáze rozvoje řetězce

Tato fáze výpočtu je zajištěna moduly memstr, param, deriv, strio, tables, product a evaluate.

Modul param

Modul param zajišťuje alokaci paměti pro záznam parametrů symbolů. K tomu poskytuje strukturu *tpmem* definovanou takto:

```
typedef struct
  { tarray heads;      /*  dynamické pole ukazatelů na paměťové bloky */
```

```
    ushort free;       /*  počet volných míst v aktuálním bloku */
    double* freeptr;  /*  ukazatel na začátek volného místa v aktuálním bloku */
    ushort lastblock;  /* počet naalokovaných bloků */
  }
tpmem;
```

Modul poskytuje funkce

InitPmem	Inicalizuje strukturu *tpmem*.
FreePmem	Reinicializuje struktru a uvolní všechny bloky včetně pole ukazatelů na ně.
ResetPmem	Reinicializuje struktru bez uvolňování bloků.
GetPmem	Vrátí ukazatel na souvislý úsek paměti do něhož lze uložit parametry symbolu, jejichž počet je dán parametrem funkce.

Modul memstr

Modul memstr zajišťuje uložení generovaného řetězce a operace potřebné pro jeho rozvíjení. Poskytuje strukturu *tmemstr*, která je definována takto:

```
typedef struct
   { int  code;            /* 0 - přepisovaný řetězec, 1 - generovaný řetězec */
     tarray   heads;      /* dynamické pole hlaviček bloků */
     tstrsym* topptr;    /* ukazatel na poslední obsazenou buňku */
     taddr    topaddr;   /* adresa poslední obsazené buňky */
     ushort valid;        /*  počet alokovaných bloků */
   }
tmemory;
```

Dynamické pole *heads* obsahuje hlavičky bloků, které jsou definovány takto:

```
typedef struct MEMHEAD
  { tstrsym* block;          /* ukazatel na paměťový blok */
    tpmem    parmem;         /* struktura pro zásnam parametrů symbolů */
    int      status;         /* stav bloku ( přítomný / smazaný ) */
    ushort lock;             /* počet zámků na bloku */
  }
tmemhead;
```

Paměť pro řetězec symbolů se tedy skládá z jednotlivých bloků a podle toho se také adresuje. Adresa buňky se skládá z čísla bloku (segment) a pořadí buňky v bloku (ofset).

V případě, že je proměnná *ShortSymbol* nastavena na 1, tj. předpokládá se neparamtrický L-systém skládá se každý blok z buněk typu *tstrsym* definovaných takto:

```
typedef struct
    { ushort code; }
tstrsym;
```

Každá buňka tedy obsahuje pouze dvojbytový kód symbolu. V opačném případě jsou buňky typu *tetxstrsym.* Tato struktura je definována následujícím způsobem.

```
typedef struct
  { ushort code;
    tinfo   info;
  }
textstrsym;
```

Kromě kódu symbolu obsahuje ještě další informaci. U parametrických symbolů je to ukazatel na pole parametrů. Na začátku a na konci každého bloku je symbol konce bloku a číslo bloku. Tyto informace umožňují sekvenční procházení pamětí oběma směry. V případě parametrického systému je pro označení konce bloku využita jedna buňka paměti. V ní je uložen kód symbolu eob (end of block) a v proměnné *info.addr.segment* příslušné buňky je pak uloženo číslo tohoto bloku. V případě neparametrického systému jsou pro tyto informace vyhrazeny na každém konci bloku dvě buňky paměti. Symboly reprezentující příkazy push a pop (uložení stavu želvičky na zásobník a vybrání stavu želvičky ze zásobníku) nemohou mít žádné parametry a proměnná *info.addr* v příslušné buňce obsahuje adresu párové závorky. V případě naparametrického systému je každý tento symbol zaznamenán ve čtyřech buňkách ve formátu

kód_symbolu segment ofset kód_symbolu

Modul obsahuje dvě proměnné typu *tmemstr* zpřístupněné ukazateli *src* a *dest*. Ukazatel *src* ukazuje na strukturu obsahující přepisovaný řetězec, ukazatel *dest* ukazuje na strukturu, do které se ukládá generovaný řetězec. Po ukončení zápisu nového řetězce je třeba volat funkci *MemWriteStop*, která smaže starý řetězec a vymění obsah ukazatelů *src* a *dest*. Předpokládá se, že v každém derivačním kroku se řetězec prodlouží, a proto se při mazání starého řetězce neuvolňují alokované hlavičky bloků ani bloky udržované strukturami *tpmem*, které se nacházejí v hlavičkách bloků. V dalším kroku jsou pak využity při vytváření nového řetězce. Počet inicializovaných hlaviček se pro každou strukturu *tmemstr* zaznamenává v proměnné *valid*.

Modul memstr je velmi specializovaný, zaměřený na co nejefektivnější provádění funkcí používaných při přepisování řetězců. Poskytuje tyto funkce:

InitMainPtr	Inicalizuje ukazatel *MainPtr* a další struktury pro procházení řetězce za účelem generování nového řetězce.

IncMainPtr	Posune ukazatel *MainPtr* na další buňku a upraví související data.
GetSrcStart	Vrací ukazatel na první buňku paměti *src*.
IncPtr	Posune ukazatel na následující buňku.
DecPtr	Posune ukazatel na předchozí buňku.
Jump	Vrací ukazatel na párový symbol push nebo pop.
MemWriteStop	Oznamuje ukončení zápisu nového řetězce. Smaže sterý řetězec a vymění obsahy ukazatelů *src* a *dest*.

„Související data“ zmiňovaná u funkcí *InitMainPtr* a *IncMainPtr* jsou struktury pro zjišťování bloků, které již nebudou při přepisování řetězce potřeba a mohou být uvolněny. Pokud L-systém neobsahuje pravidla s levým kontextem, může být každý blok uvolněn v okamžiku, kdy ho opustí ukazatel *MainPtr*. V opačném případě se udržuje struktura *MaxLC* a zásobník *maxlcstk*, pomocí kterých se určuje, který blok již může být uvolněn. K tomuto účelu se používají také proměnné *lock* v hlavičkách bloků.

Modul deriv

Modul deriv poskytuje funkci *DeriveStep*, která provede jeden derivační krok nad řetězcem uloženým v paměti *src* v modulu memstr. Tato funkce dostává jeden parametr typu long int, který určuje aktivní a neaktivní tabulky pravidel v tomto derivačním kroku.

Modul strio

Poskytuje funkce pro zápis řetězců do souborů v textovém nebo binárním formátu.

Program draw

Program draw interpretuje metodou 3D želvičky řetězce generované programem rewrite. K reprezentaci a zobrazování 3D objektů používá struktury a funkce knihovny Iris Inventor, knihovny X Toolkit a Motif a moduly targ, t3d, viewer a filetab.

Využití knihovny Inventor v programu draw

Objekt je ve vnitřním formátu Iris Inventor reprezentován stromem jehož uzly představují jednak základní tvary, z nichž se objekt skládá (tělesa, lomené čáry, množiny polygonů, NURBS křivky a plochy a jiné), jejich vlastnosti (např. materiál, způsob kreslení, normálové vektory, textury, transformace atd.), skupinové vrcholy (tj. vrcholy, které mohou mít syny a mohou tedy být kořenem netriviálního podstromu celé reprezentace) a

další typy uzlů, které však nejsou programem draw přímo využívány pro generování 3D objektů, a proto je zde nebudeme podrobněji popisovat. Tyto uzly však mohou být obsaženy ve stromech reprezentujících předdefinované objekty a mohou tedy být do stromu vloženy příslušnou akcí želvičky.

Při zobrazování objektu se strom prochází do hloubky a každý uzel reprezentující tvar je zobrazen. Uzly reprezentující vlastnosti objektů ovlivňují způsob zobrazení. Například uzel SoLineSet se odkazuje do aktuálního seznamu vrcholů, aby určil vrcholy lomené čáry, dále do aktuálního seznamu materiálů a pro kreslení čar používá aktuální styl kreslení. Tyto uzly reprezentují vlastnosti kreslených objektů až do doby než je ve stromu nalezen další uzel stejného typu. Výjimku tvoří uzly reprezentující geometrické transformace. Transformace nenahrazují jedna druhou, ale skládají se postupně tak, jak jsou jednotlivé uzly transformací ve stromě nacházeny.

Následující tabulka obsahuje typy uzlů používaných při generování objektů programem draw.

Jméno uzlu	Popis	Využití
SoCoordinate3	Uzel definující aktuální seznam vrcholů.	Uchovává vrcholy pro definice lomených čar a polygonů.
SoLineSet	Uzel reprezentující lomenou čáru určenou aktuálním seznamem vrcholů.	Reprezentace souvislých množin segmentů generovaných příkazy F, a G.
SoFaceSet	Uzel reprezentující množinu polygonů určených aktuálním seznamem vrcholů.	Reprezentace polygonů.
SoMaterial	Uzel definující aktuální seznam materiálů.	Využívá se pro určení barvy segmentů a polygonů.
SoTexture2	Uzel definující aktuální texturu.	Používá se při vkládání textur.
SoNormal	Uzel definující aktuální seznam normálových vektorů.	Definuje normálové vektory polygonů.
SoDrawStyle	Uzel definující styl kreslení objektů.	Používá se pro určení tloušťky čar.

SoMatrixTransform	Transformace	Používá se pro správné zorientování vkládaných předdefinovaných objektů a pro posunutí při současném zobrazení více objektů.
SoEnviroment	Uzel definující optické vlastnosti prostředí.	Definuje intenzitu rozptýleného světla.
SoSeparator	Skupinový uzel. Při kreslení objektu se při přechodu přes tento uzel uloží aktuální vlastnosti objektů (materiály, textura, normálové vektory, transformace) na zásobník a při návratu z podstromu se opět vyberou.	Uzel SoSeparator je kořenem každé větve omezené symboly '[' a ']'.
SoSwitch	Tento uzel může mít několik synů a určuje zda při procházení stromem budou navštíveni všichni, žádný z nich nebo jeden vybraný.	Uzel tohoto typu je kořenem celého stromu. Využívá se pro animaci.
SoShapeHints	Tento uzel poskytuje dodatečné informace pro vykreslování polygonů (např. zda je těleso uzavřené, jaké je pořadí vrcholů v polygonech při pohledu zvnějšku tělesa apod.)	Informuje, že těleso není uzavřené a nelze tedy vypustit odvrácené polygony.

Pro zobrazování objektů se používá třída *SoExaminerViewer.* Tato třída volá funkce knihoven X Toolkit a Motif pro vytvoření a správu okna, do ve kterém pak objekt zobrazuje. Pro nakreslení objektu se volají funkce knihovny Open GL, která na počítačích Silicon Graphics využívá jejich hardwarovou podporu 2D a 3D grafiky.

Modul targ

Modul targ pracuje stejným způsobem jako modul rarg programu rewrite. Čte argumenty programu a podle jejich hodnot nastaví příslušné globální proměnné.

Modul filetab

Tento modul implementuje struktury pro reprezentaci asociačních souborů pro textury a vkládané objekty. Poskytuje funkce pro načtení asociačního souboru a vyhledání souborů asociovaných s daným znakem abecedy. Obsah asociačního souboru je reprezentován hašovací tabulkou do níž se vkládají struktury typu *tfiletabentry*, které jsou definovány takto:

```
typedef struct
  { struct CONNECT *next;
    tid symbolname;
    tarray filenames;
  }
tfiletabentry;
```

Dynamické pole *filenames* pak udržuje ukazatele na řetězce obsahující jména asociovaných souborů.

Modul t3d

Tento modul přímo zajišťuje interpretaci řetězce. Nejprve je načtena hlavička vstupního souboru a podle ní inicializováno pole *SwitchArray* tak, že pro každý kód symbolu obsahuje ukazatel na funkci zajišťující odpovídající akci. Modul obsahuje tři hlavní struktury používané pro reprezentaci stavu želvičky, stavu právě vytvářené lomené čáry a stavu aktuálního polygonu. Jsou to struktury:

```
class turtle3D
  { public:
      SbMatrix Dir;            /* orientace želvičky - vektory H, L a U */
      SbVec3f  Pos;            /* pozice želvičky */
      SbVec3f  Head;           /* vektor H */
      SbColor  color;          /* aktuální barva kreslení */
      double   width;          /* aktuální šířka čáry */
      turtle3D ( turtle3D& );  /* kopírovací konstruktor */
      turtle3D() {};           /* konstruktor */
  };

typedef  struct
  { SoSeparator   *Parent;        /* ukazatel na otcovský uzel */
    SoCoordinate3 *LineCoord;     /* uzel obsahující body určující lomenou čáru */
    unsigned PointCount;          /* počet uzlů lomené čáry */
    int MatInserted;              /* byl již vložen uzel SoMaterial ? */
    int DrawStyleInserted;        /* byl již vložen uzel SoDrawStyle ? */
    int AnyInsert;                /* byl vložen nějaký  uzel ? */
    int Line;                     /* počet vrcholů ležících v jedné přímce */
```

```
    ushort Counter;                 /* čítač používaný při potlačování přímých linií  */
  }
tbranchstatus;


typedef struct
  { SoCoordinate3 *PolyCoord;  /* uzel obsahující vrcholy polygonu */
    ushort PointCount;              /* počet vrcholů polygonu */
  }
tpolystatus;
```

Třída *turtle3D* reprezentuje stav želvičky. Kvůli zrychlení vykonávání příkazů ‘F’, ‘f’, ‘G’ a ‘g’ je vektor **H** uchováván v proměnné *Head* ačkoli lze jeho hodnotu zjistit z matice *Dir* reprezentující orientaci želvičky.

Struktura *tbranchstatus* reprezentuje stav právě vytvářené lomené čáry. Všechny uzly reprezentující tuto čáru a její vlastnosti budou do stromu vloženy jako synové uzlu, jehož adresu obsahuje ukazatel *Parent.* Ukazatel *LineCoord* obsahuje adresu uzlu, do kterého jsou vkládány jednotlivé body lomené čáry. Proměnné *MatInserted* a *DrawStyleInserted* zaznamenávají, zda je třeba pro aktuální lomenou čáru vložit nový uzel typu *SoMaterial* resp. *SoDrawStyle*, protože aktuální čára má jiné vlastnosti než byly definovány v naposledy vložených uzlech těchto typů. Proměnná *AnyInsert* zaznamenává, zda otcovský uzel má alespoň jednoho syna. Pokud ne, může být po ukončení záznamu lomené čáry zrušen. Tato situace může nastat pokud lomená čára neobsahuje ani jeden segment. Proměnná *Line* zaznamenává, zda právě vkládaný uzel leží v jedné přímce s posledními dvěma. Pokud ano, poslední bod je odstraněn a nahrazen tímto novým bodem. Tím se snižuje počet segmentů lomené čáry. Proměnná *Counter* se používá při potlačování přímých linií k zaznamenávání počtu segmentů vložených od posledního náhodného pootočení želvičky. Body lomené čáry jsou do struktury zaznamenávány při provádění příkazů ‘F’ a ‘G’. Záznam lomené čáry je ukončen při pohybu želvičky bez kreslení, tj. při vykonávání příkazů ‘f’ a ‘g’ a při změně vlastností kreslené čáry (barva, tloušťka). Po ukončení záznamu jsou příslušné uzly vloženy do stromu a struktura *tbranchstatus* je znovu inicializována.

Struktura *tpolystatus* reprezentuje právě vytvářený polygon. Polygon je ve stromě reprezentován uzly typu *SoMaterial*, *SoNormal*, *SoCordinate3* a *SoFaceSet*, které se do stromu vkládají jako synové uzlu určeného ukazatelem *Parent* v aktuální struktuře *tbranchstatus*.

Pro každou ze tří právě popsaných struktur existuje jeden zásobník. Zásobníky pro *turtle3D* a *tbranchstatus* jsou používány při vykonávání příkazů ‘[‘ a ‘]’, zásobník pro struktury *tpolystatus* se používá při vykonávání příkazů ‘{‘ a ‘}’.

Modul obsahuje funkci *ReadStr*, která čte ze vstupního souboru kódy symbolů a pro každý vyvolá přes pole *ArraySwitch* odpovídající akci. Pro symboly, které želvička ignoruje, se volá prázdná akce.

Modul viewer

Modul viewer poskytuje funkci *ViewTree*, která zobrazí objekt vytvořený modulem t3d. K zobrazení používá objekt *SoXtExaminerViewer* definovaný v knihovně Inventor. V případě požadavku na animaci vývojové sekvence vloží do rámečku okna, ve kterém se objekt zobrazuje, tlačítka pro řízení animace.

Uživatelské rozhraní aplikace - program lsdraw

Uživatelské rozhraní je vytvořeno pomocí knihovny X Toolkit a její nadstavby, knihovny Motif. Jednotlivá okna jsou definována ve zvláštních souborech v jazyce UIL (User Interface Language). Kompilací těchto souborů vzniknou tzv. uid soubory, ze kterých je program schopen načítat za běhu definice jednotlivých oken pomocí funkcí poskytovaných knihovnou Motif.

Program lsdraw se skládá z modulů larg, ui, data a cmdline. Modul larg zajišťuje přečtení parametrů programu stejným způsobem jako moduly rarg a targ v programech rewrite a draw.

Modul ui

Modul ui zajišťuje inicializaci aplikace, načtení oken uživatelského rozhraní z uid souborů, definuje „callback“ funkce pro jednotlivé události. Dále definuje struktury pro uchování hodnot všech parametrů programů rewrite a draw tak, jak jsou zadány v dialogových oknech uživatelského rozhraní.

Modul data

Modul data poskytuje struktury a funkce pro přenášení dat mezi dialogovými okny a programem, ukládání těchto dat na disku a jejich čtení z disku.

Modul cmdline

Modul poskytuje globální pole znaků *CmdLine* a funkci *CreateCmdLine*, která na základě svých parametrů vytvoří příkaz pro shell, který spouští kolonu rewrite | draw s parametry zadanými v dialogových oknech uživatelského rozhraní a tento příkaz uloží do pole *CmdLine.*

Některá možná rozšíření aplikace

Ačkoli je aplikace určena především pro modelování rostlin a poskytuje k tomu potřebné prostředky, nelze ani pomocí těchto prostředků jednoduchým způsobem generovat zcela realistické obrazy rostlin. Problém spočívá především v generování a zobrazování ploch. Při interpretaci řetězce vytváří želvička jednotlivé polygony zcela nezávisle na sobě. Při zobrazování pak chybí informace o sousedních polygonech v téže ploše a nelze použít stínování vyhlazující hrany mezi polygony, např. metodu aproximace normály. Tento problém je možné řešit následující modifikací interpretace symbolu '.' .

L-systém musí být navržen tak, měl aby symbol '.' při generování téhož vrcholu pro různé polygony vždy stejný parametr. Při interpretaci řetězce jsou pro každou hodnotu parametru zaznamenávány polygony, které obsahují vrchol generovaný příkazem '.' s tímto parametrem. Pak lze snadno vypočítat normálu plochy v tomto bodě jako průměr normál polygonů obsahujících tento vrchol a při zobrazování použít stínování metodou aproximace barvy nebo normály.

Další možností rozšíření je generování sítě řídících vrcholů ploch obdobným způsobem jako vrcholů polygonů. Symbol '{' by například mohl mít tři parametry, přičemž první by určoval typ plochy a další by udávaly velikost sítě. Takto generované plochy by měly, na rozdíl od předdefinovaných ploch vkládaných ve fázi interpretace řetězce, možnost vývoje pomocí přepisovacích pravidel L-systému.

Závěr

Aplikace lsdraw, jejíž vytvoření bylo hlavním cílem této práce, podporuje standardní metody využívání L-systémů v počítačové grafice. Nabízí dále běžné prostředky pro modelování vývoje rostlin uvedené např. v [5] a navíc např. pokrývání ploch texturami a randomizaci interpretace řetězce. Nebylo zcela dosaženo možnosti generovat realistické obrazy rostlin, byla však navržena rozšíření aplikace, pomocí kterých by bylo možno tohoto cíle dosáhnout. Tyto návrhy nebyly implementovány z časových důvodů.

Implementace na počítači Silicon Graphics umožňuje díky velké výpočetní síle zejména v oblasti grafiky generovat i poměrně složité objekty skládající se z velkého počtu polygonů. Nevýhodou pak je nepřenositelnost na platformy, kde nejsou implementovány použité knihovny, zejména Iris Inventor a Open GL.

Literatura

[1] A. Lindenmayer : *Mathematical models for cellular interaction in development*, Part and II. *Journal of Theoretical Biology*, 18:280 - 315, 1968.

[2] A. Lindenmayer : *Adding continuous components to L-systems*. In G. Rozenberg and A. Saloma, editors, *Lecture Notes in Computer Science* 15, pages 53 - 68. Springer-Verlag, Berlin, 1974.

[3] P. Prusinkiewicz, J. Hanan : *Lindenmayer Systems, Fractals and Plants*, volume 79 of *Lecture notes in Biomathematics*. Springer-Verlag, Berlin, 1989.

[4] P. Prusinkiewicz, J. Hanan : *Visualization of botanical structures and processes using parametric L-systems*. In D. Thalman, editor, *Scientific visualization and Graphics Simulation*, pages 183 - 201, J. Wiley & Sons, 1990

[5] P. Prusinkiewicz, A. Lindenmayer : *Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[6] J. Žára a kol. : *Počítačová grafika - principy a algoritmy.* Grada, Praha, 1992