

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Generování rostlin pomocí L-systémů

DIPLOMOVÁ PRÁCE

Marek Pasičnyk

Brno, jaro 2010

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Vít Kovalčík, Ph.D.

Poděkování

Chtěl bych poděkovat

Shrnutí

Tato práce se zabývá problematikou Lindenmayerových systémů, jakožto nástroje pro generování rostlin. Rozebírá pak také metody a implementaci L-systémů a jejich grafickou interpretaci. Součástí je i zásuvný modul pro generování rostlin pro systém Vrecko.

Klíčová slova

Lindenmayerovy systémy, L-systémy, generování rostlin, generování stromů, Vrecko, OpenSceneGraph, LSystem

Obsah

1	Úvod	1
2	Lindenmayerovy systémy	3
2.1	<i>Vývoj rostlin pomocí procesu přepisování</i>	3
2.2	<i>D0L-systémy</i>	4
2.3	<i>L-systémy se závorkami</i>	5
2.4	<i>Kontextové L-systémy</i>	6
2.4.1	Párování levého kontextu	7
2.4.2	Párování pravého kontextu	8
2.5	<i>Parametrické L-systémy</i>	8
2.6	<i>Stochastické L-systémy</i>	9
3	Modelování rostlinných organismů	12
3.1	<i>Interpretace želví grafikou</i>	12
3.1.1	Planární želví grafika	12
3.1.2	Interpretace závorkových L-systémů	13
3.1.3	Rozšíření pro interpretaci L-systémů v 3D	13
3.2	<i>Subapikální L-systémy</i>	14
3.3	<i>Vzory větvených struktur</i>	14
3.3.1	Monopodiální vzor	14
3.3.2	Sympodiální vzor	15
3.4	<i>Vlastnosti větvených struktur rostlinných organismů</i>	16
4	Zásuvný modul LSystem	18
4.1	<i>Vývoj zásuvného modulu</i>	19
4.2	<i>OpenSceneGraph</i>	19
4.3	<i>Vrecko</i>	19
5	Generování slov pro modely rostlin	20
5.1	<i>Soubory L-systémů</i>	20
5.1.1	Formát LS	22
5.1.2	Formát XML	22
5.2	<i>Řetězce modulů</i>	23
5.2.1	Vnitřní struktura řetězce	24
5.2.2	Inicializace	24
5.2.3	Připojování řetězců	25
5.2.4	Přístup k datům řetězce	25
5.3	<i>Implementace přepisovacích pravidel</i>	26
5.4	<i>Zpracování L-systémů</i>	27
5.4.1	Výběr nevhodnějšího L-systému	28
5.4.2	Inicializační fáze	28
5.4.3	Přepisovací fáze	29
5.5	<i>Rozdelení L-systémů</i>	30
5.5.1	D0L-systémy	30

5.5.2	Parametrické stochastické bezkontextové L-systémy	30
5.5.3	Parametrické stochastické kontextové L-systémy	31
5.6	<i>Dotazy</i>	33
5.7	<i>Podřízené L-systémy</i>	35
6	Interpretace slov	37
6.1	<i>Spojení s grafem scény</i>	37
6.2	<i>Interpretace pomocí želví grafiky</i>	38
6.2.1	Zásobník pro ukládání želvích instancí	38
6.2.2	Želví rozhraní	39
6.2.3	Želví příkazy	39
6.3	<i>Generování geometrie</i>	39
6.3.1	Válce s klobuby	40
6.3.2	Spojité válce	40
6.3.3	Obdélníky	41
6.4	<i>Textury</i>	42
6.4.1	Mapování textur na navazující válce	42
6.4.2	Mapování textur na obdélníky	43
6.5	<i>Minimalizace příčného náklonu</i>	43
6.6	<i>Odezva na směrové podněty</i>	44
6.6.1	Geotropismus	45
6.6.2	Diatropismus	46
6.7	<i>Fluktuace úhlů</i>	47
7	Výkonnostní testy modulu LSystem	50
8	Závěr	52
	Literatura	54
A	Přehled interpretovaných příkazů pro želví grafiku	55
A.1	<i>Rotace želvy</i>	55
A.2	<i>Změna vlastností želvy</i>	56
A.3	<i>Změna pozice želvy</i>	56
A.4	<i>Operace na zásobníku želv</i>	56
A.5	<i>Změna nastavení tropismu</i>	57
B	Přehled parametrů pro nastavení L-systémů	58
B.1	<i>Parametry generátoru slov</i>	58
B.2	<i>Parametry želvy</i>	58
B.3	<i>Parametry vykreslované geometrie</i>	59
B.4	<i>Tropismus</i>	60
B.5	<i>Geometrie pro ladění</i>	61
C	Přiložené CD	62

Kapitola 1

Úvod

Snaha převést přírodu do digitální podoby provází počítačovou grafiku již od samého počátku. Po celou tuto dobu je zpracování přírodních procesů jednou z nejvíce studovaných a tedy i rozvíjejících se oblastí. Od počátku totiž byly přírodní procesy příliš složité, než aby bylo možné vytvořit je pomocí počítače bez jakéhokoli zpracování a zjednodušení. Zároveň jsou však součástí témař každé animace, filmu, simulace či počítačové hry. Díky tomu vzniklo za posledních padesát let od vynálezu světelného pera Ivana Sutherlanda, tedy od počátku moderní počítačové grafiky, obrovské množství více či méně úspěšných metod. Proto také dnes ty nejmodernější metody umožňují někdy i fotorealistické zpracování modelů nebo simulaci různých přírodních jevů graficky i fyzikálně. S určitými omezeními je to navíc možné i v reálném čase. V tomto vývoji samozřejmě hrál velkou roli i vývoj samotné počítačové techniky.

První pokusy o vytvoření počítačových modelů rostlinných organismů pochází z roku 1966, kdy Stanislaw Ulman přišel s prvním mechanismem pro vytváření větvených struktur. Využil k tomu principu von Neumannova celulárního automatu. Zanedlouho však Aristid Lindenmayer publikoval koncept Lindenmayerových systémů, které se později staly nejpoužívanější metodou pro generování rostlinných organismů. Během posledních 45 let se z generování rostlin stal jeden z důležitých mezivědních oborů. Na jedné straně vždy stáli biologové, jež se za pomocí počítačových modelů a simulací snažili pochopit principy vývoje a stavby rostlin. Na druhé straně jsou pak specialisti na počítačovou grafiku, jež stromy používají jako součást grafických scén. Kombinací poznatků obou vědních oborů tak vznikají stále nové metody. Výsledkem je, že modely rostlin jsou dnes často těžko rozeznatelné od svých opravdových protějšků.

Tato diplomová práce se věnuje modelování rostlinných organismů za použití Lindenmayerových systémů. Cílem bylo navrhnout zásuvný modul LSystem pro systém Vrecko. Tento modul umožňuje na základě uživatelsky definovaných parametrů vytvořit širokou škálu různých modelů. Implementován totiž není pouze původní Lindenmayerův koncept, ale také řada metod, jež jej rozšiřuje o další možnosti. Modely tak mohou například reagovat na okolní prostředí. Lindenmayerovy systémy lze díky jejich všeobecnosti použít i pro vytváření různých neorganických modelů. Většina zde zmíněných metod je použitelná pro vytváření libovolného rostlinného organismu. Při implementaci modulu LSystem však byl kladen důraz na vytváření modelů stromů a keřů, jež pak mohou být použity pro vytváření venkovních scén v systému Vrecko.

Začátek této diplomové práce se věnuje především samotným Lindenmayerovým systé-

mům, jejich typům a možnostem. Třetí kapitola pak rozebírá problematiku modelování rostlinných organismů. Zbývající kapitoly se již věnují samotnému návrhu a implementaci zásvného modulu LSystem. Čtvrtá kapitola se zabývá obecným popisem modulu. Následující pátá kapitola pak obsahuje popis implementace generování řetězců pomocí L-systémů. Interpretace řetězců a vytváření samotné geometrie modelů je pak obsahem šesté kapitoly. Práci uzavírá sedmá kapitola shrnující výkonnostní i zobrazovací výsledky.

Kapitola 2

Lindenmayerovy systémy

Lindenmayerovy systémy, zkráceně L-systémy, byly vytvořeny jako matematická teorie pro generování rostlin [9]. Původně byl kladen důraz spíše na obecnou topologii. Byly totiž vytvořeny pro simulaci vývoje větších částí rostlin nebo buněk mnohobuněčných organismů. Geometrie ani podrobnější detaily v této původní teorii zahrnuty nebyly. Později se však objevilo několik geometrických interpretací L-systémů, díky nimž se staly L-systémy univerzálním nástrojem pro simulaci a modelování rostlin. V této práci je například použito modelování pomocí želví grafiky.

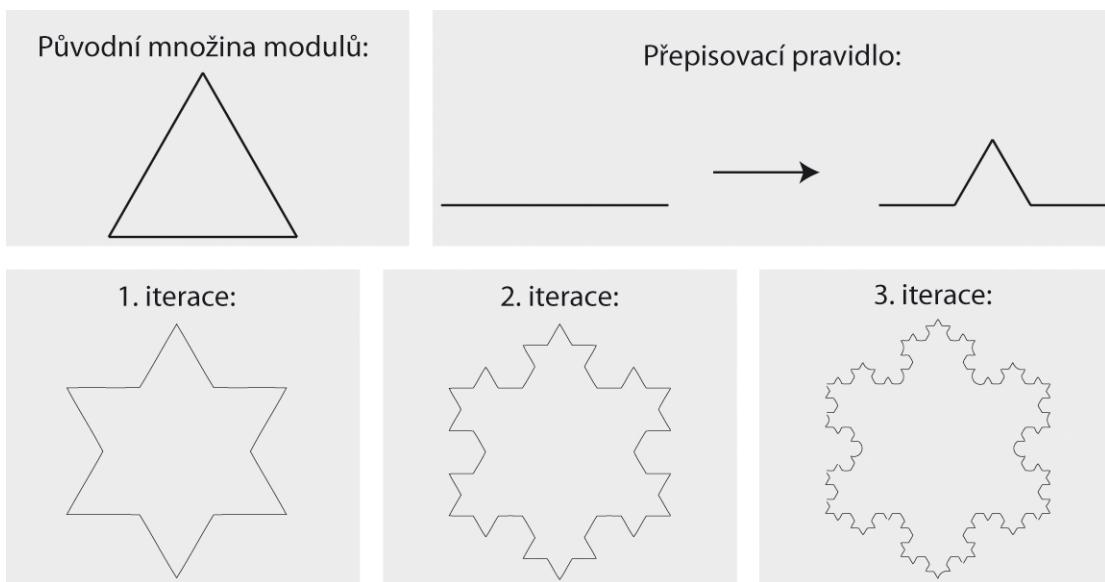
Lindemayerovy systémy však nenalezly uplatnění zdaleka jen v generování rostlinných organismů. Existuje řada projektů, jež různými způsoby experimentuje s použitím L-systémů.

2.1 Vývoj rostlin pomocí procesu přepisování

Jádro L-systémů spočívá v použití přepisovacího systému. Jedná se o opakování nahrazování modulů předchůdců pomocí sady přepisovacích pravidel jejich následníky, přičemž následník je řetězec modulů. Všechny moduly náleží do konečné abecedy modulů. Lze tak z jednoduchého původního objektu vytvořit opakováním přepisovacího procesu komplexní model. Modulem je myšlen libovolný objekt, jež většinou reprezentuje nějakou část simulovaného modelu nebo jeho vlastnost. Při použití pro simulaci biologických procesů se tak může jednat například o buňky nebo různé typy rostlinných orgánů.

Jedním z prvních grafických modelů, používajících přepisovací pravidla, byla Kochova křivka, případně Kochova vločka [8]. Šlo o jednoduchý bezkontextový přepisovací systém, který obsahoval pouze jediné pravidlo přepisující jeden grafický prvek na jiný (2.1). Později tento model podstatně rozšířil Benoît Mandelbrot, jež přidal systémy podporující přepisování úseček o různých délkách a hlavně podporu větvených topologií.

Největší pozornost však byla ubírána ke studiu systémů založených na přepisování řetězců znaků. Velkým přínosem v tomto odvětví byly na konci padesátých let 20. století Chomského formální gramatiky, které využívaly princip přepisování pro popis syntaxe přirozeného jazyka. V roce 1968 pak biolog Aristid Lindenmayer představil odlišný typ mechanismu přepisujícího řetězce. Tento mechanismus byl posléze pojmenován jako L-systémy. Zásadní odlišnost L-systémů od Chomského gramatik spočívá v použití přepisovacích pravidel. Zatímco v Chomského gramatikách jsou přepisovací pravidla aplikována postupně, v L-systémech jsou použita paralelně v jednom derivačním kroku na všechny symboly přepisovaného řetězce. Motivací pro tento přístup byla podobnost s biologickými procesy. Pří-



Obrázek 2.1: Kochova vločka a její první tři iterace přepisovacího procesu.

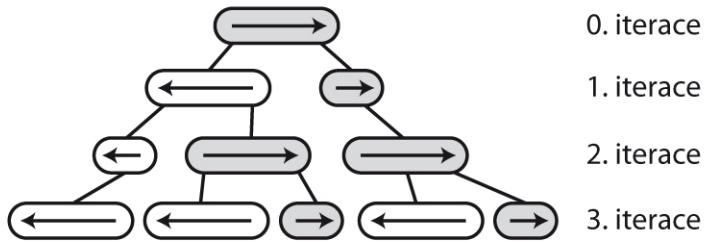
kladem mohou být mnohobuněčné organismy, ve kterých se dělí všechny buňky současně. Na rozdíl od přírodních pochodů vývoj L-systémů probíhá v diskrétních krocích. Kvůli reprezentaci modelů jako řetězce znaků se často místo pojmu modul používá pojed symbol.

2.2 D0L-systémy

Deterministické bezkontextové L-systémy, zkráceně D0L-systémy, jsou nejjednodušší formou L-systémů. Původní Lindenmayerovy systémy zahrnovaly právě pouze tento typ. Takovýto typ L-systému se skládá z abecedy modulů, přepisovacích pravidel a axiomu, jež slouží jako počáteční řetězec modulů. Na obrázku 2.2 je znázorněna simulace vývoje mnohobuněčného vlákna buněk nacházející se v bakterii *Anabaena catenula* [11]. Tyto řetězce jsou tvořeny dvěma typy buněk. Prvním typem jsou mladé, kratší buňky. Druhý typ jsou buňky starší a delší. Každá buňka má také svou polaritu, která určuje směr jejich růstu.

Formální definice D0L-systémů je následující [15]:

- Abeceda V je konečná množina *symbolů*.
- Slovo je posloupnost symbolů nad abecedou V . Množina všech těchto posloupností se označuje jako V^* .
- Přepisovací pravidlo je uspořádaná dvojice (a, u) zapsaná jako $a \rightarrow u$, kde a je symbol náležící V a u je slovo náležící V^* . Symbol a se nazývá *předchůdce* a slovo u *následník*.


 Obrázek 2.2: První tři iterace vlákna buňky *Anabaena catenula*

- DOL-systém je uspořádaná trojice $G = \langle V, \omega, P \rangle$, kde V je abeceda, $\omega \in V^*$ je počáteční slovo nazývané axiom a P je množina přepisovacích pravidel takových, že pro $\forall a \in V : \exists p_a \in P$, kde p_a označuje přepisovací pravidlo, jehož předchůdce je modul a .

Dle obecných konvencí bývá přepisovací pravidlo $a \rightarrow u$ označováno jako odpovídající modulu a a pro modul, kterému neodpovídá žádné z přepisovacích pravidel, bývá použito pravidlo identity $a \rightarrow a$. Modul je možné během vývoje L-systému odstranit pomocí použití ϵ pravidla.

Dle této definice je pak možné pro vývoj vláken bakterie *Anabaena* zavést L-systém, jež popisuje jeho vývoj. V těchto pravidlech nahrazují delší buňky moduly a a kratší buňky moduly b . Index u modulů označuje jejich orientaci.

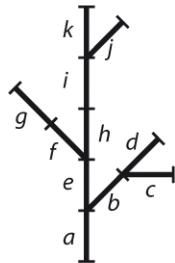
$$\begin{aligned}\omega &: a_p \\ p_1 &: a_p \rightarrow a_l b_p \\ p_2 &: a_l \rightarrow b_l a_p \\ p_3 &: b_p \rightarrow a_p \\ p_4 &: b_l \rightarrow a_l\end{aligned}$$

2.3 L-systémy se závorkami

Obecné DOL-systémy poskytují možnosti pouze pro vytvoření sekvence modulů. Aby však bylo možné vytvořit slova reprezentující rostliny, je nutné umožnit L-systémům vytvářet větvené topologie. Pro reprezentaci větvených struktur je potřeba zavést dva moduly. Tyto moduly byly již součástí původního Lindenmayerova konceptu [9]. Pro označení začátku a konce vedlejší větve se používají symboly pravé a levé hranaté závorky.

Slova, vyprodukovaná derivacemi přepisovacích pravidel závorkového L-systému, se označují jako stromy. Příklad takového stromové struktury vytvořené L-systémem se závorkami je na obrázku 2.3. Jedná se o základní větvenou strukturu s jednou hlavní osou tvořenou moduly $aehik$. Slovo reprezentující tento strom lze pomocí jednoho řetězce zapsat jako $w = a[b[c]d]e[fg]hi[j]k$.

Řetězce modulů a , e a hi se označují jako meziuzly a řetězce $b[c]d$, fg a j jako vedlejší větve. Modul k je hrotom hlavní osy. Obdobně lze pojmenovat i moduly na nižších úrovních



Obrázek 2.3: První tři iterace vlákna buňky *Anabaena catenula*

a označit tak například moduly c, d, g a j jako hroty vedlejších větví.

Strom lze podle [13] definovat jako slovo w nad abecedou $V_E = V \cup \{[,]\}$ pro které platí, že

- $w = x_1[a_1]x_2[a_2]\dots x_n[a_n]x_{n+1}$,
- podslova $x_1, x_2, \dots, x_{n+1} \in V^*$ neobsahují moduly závorek a
- podslova $a_1, a_2, \dots, a_n \in V_E^*$ jsou stromy.

2.4 Kontextové L-systémy

Přepisování pravidel u 0L-systémů je bezkontextové. Při produkci následníků tedy nezáleží na modulech v okolí předchůdce. Při přepisování kontextových L-systémů však musí kromě předchůdců odpovídat také jejich sousední moduly. Tímto způsobem mohou mezi sebou jednotlivé moduly komunikovat nebo posílat ostatním modulům signály. Lze tak simuloval reakci na proudění látek rostlinou nebo informovat moduly o globálních vlastnostech celého L-systému, jako je délka stonku, počet květů, stáří apod. Moduly však takto mohou komunikovat i s okolím L-systému a získávat tak informace například o množství světla, množství živin, ročním období nebo překážce v růstu.

Existuje mnoho implementací kontextových L-systémů. Nejpoužívanější jsou 1L-systémy a 2L-systémy. U 1L-systémů je brán zřetel buď pouze na levý nebo pouze na pravý kontext modulu předchůdce. Zápis pravidel s jednostranným kontextem tedy může být ve tvaru

$$lk < \text{předchůdce} \rightarrow \text{následník nebo předchůdce} > pk \rightarrow \text{následník},$$

kde lk je levý a pk pravý kontext. Obecnější třídu tvoří (k,l)-systémy, kde levý kontext tvoří slovo o délce k modulů a pravý kontext slovo o délce l modulů. V některé literatuře bývají označovány jako IL-systémy. V této práci se zabývám jejich podmnožinou, třídou 2L-systémů. Přepisovací pravidla v tomto případě zjišťují levý i pravý kontext s maximální

2.4. KONTEXTOVÉ L-SYSTÉMY

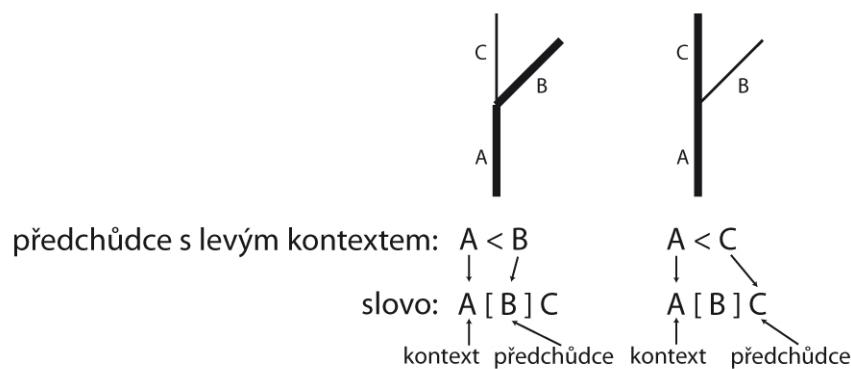
délkou 1. Touto třídou lze pokrýt všechny požadované vlastnosti L-systémů na komunikaci modulů mezi sebou i s okolím. Pravidla 2L-systémů se zapisují v následující podobě:

Ik < předchůdce > pk → následník.

U bezzávorkových L-systémů je proces zjišťování, zda přepisovací pravidlo odpovídá kontextu předchůdce, triviální a jednoznačný. Pro závorkové systémy se však literatura rozchází a jsou použity rozdílné podmínky. Jde především o zpracování pravého kontextu. Pro tento práci jsem se rozhodl použít způsob, jež obdobně používá ve své práci James Scott Hannan [5]. Na rozdíl od jiných přístupů respektuje topologickou strukturu rostliny. Pokud totiž algoritmus například při zpracování pravého kontextu narazí na větvení, porovnává s pravým kontextem každou větev. U jiných přístupů se někdy porovnává pouze větev, která s předchůdcem sousedí v řetězci. Příklad je zobrazen na obrázku 2.5.

2.4.1 Párování levého kontextu

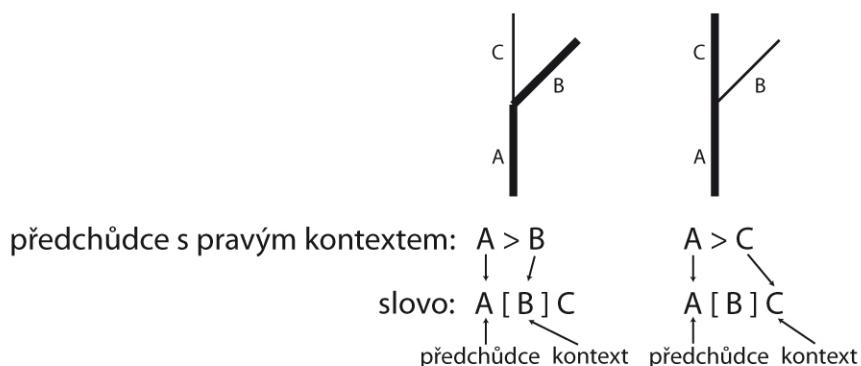
Při párování levého kontextu dochází k porovnání modulů kontextu s moduly, které jsou v řetězci slova umístěny nalevo od modulu předchůdce. V topologii rostlinného organismu se tedy jedná o části rostliny, jež leží na nejkratší cestě ke kořenům. Pro každý modul existuje pouze jedna taková cesta a párování se tedy provádí pouze s jedním řetězcem modulů. Tato cesta, jež je v topologii spojitá, nemusí v řetězci slova tvořit spojitý řetězec modulů. Pokud je při tomto párování nalezen modul], znamená to, že zde končí vedlejší větev. Ta však topologicky nenavazuje a musí být přeskočena. Párování tedy pokračuje za odpovídajícím symbolem [. Obrázek 2.4 ukazuje, jakým způsobem se levý kontext páruje. Modul A je zde levým kontextem pro předchůdce B i C. Modul B však není předchůdcem C, neboť spolu topologicky nijak nesousedí.



Obrázek 2.4: Párování levého kontextu.

2.4.2 Párování pravého kontextu

Párování pravého kontextu porovnává moduly napravo od předchůdce. Jde tedy o moduly, jež vyrůstají z předchůdce. Situace je zde komplikovanější, neboť je někdy zapotřebí kvůli větvení provést párování na více řetězcích. Pokud totiž za předchůdcem následuje rozvětvení symbolizované modulem `[`, je nutné provést párování s hlavní větví a se všemi vedlejšími větvemi. Pokud alespoň v jednom případě dojde ke shodě, je pravidlo považováno za odpovídající danému modulu předchůdce. Na obrázku 2.5 je znázorněno jednoduché větvení, kdy ke shodě dochází v případě, že pravý kontext bude modul *B* nebo *C*.



Obrázek 2.5: Párování pravého kontextu.

2.5 Parametrické L-systémy

Přestože doposud popisované typy L-systémů poskytují širokou škálu možností pro generování rostlin, jejich možnosti jsou v jistém směru omezené. Problém spočívá v tom, že veškeré moduly lze použít pouze v celočíselných násobcích. Výsledkem pak je, že nelze například vytvořit tak jednoduchý tvar, jako je rovnoramenný trojúhelník, jehož základna má v poměru k ramenům iracionální délku. Pro odbourání těchto problémů je nutné zavést parametrické L-systémy, jež pracují nad parametrickými slovy. Parametrická slova se skládají z modulů, které kromě znaku obsahují také přidružené parametry.

Identifikující znak *A* náleží do abecedy *V* a parametry a_1, a_2, \dots, a_n náleží do množiny \mathfrak{R} . Takovýto rozšířený modul se značí jako $A(a_1, a_2, \dots, a_n)$ a patří do množiny $M = V \times \mathfrak{R}^*$, kde \mathfrak{R}^* je množina všech konečných posloupností parametrů.

Pravidla parametrických 0L-systémů mají tvar

předchůdce (parametry) : podmínka → následník.

Parametry modulů následníka nemusí mít nutně tvar reálného čísla. Typicky jde o arit-

2.6. STOCHASTICKÉ L-SYSTÉMY

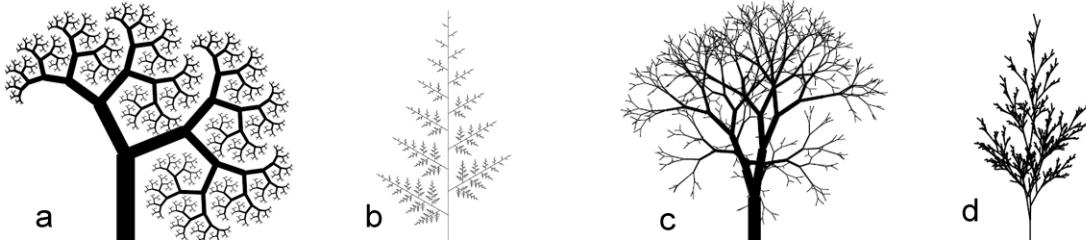
metické výrazy, které se před každým derivačním krokem vyhodnotí na základě hodnot formálních parametrů předchůdce. Tyto parametry jsou označovány jako Σ a aritmetické výrazy, jež tyto parametry používají $E(\Sigma)$. Dalším rozšířením parametrických systémů je podmínka. Přepisovací pravidlo může být použito pro přepis pouze tehdy, pokud je podmínka splněna. Jedná se o logický výraz označovaný jako $C(\Sigma)$. Podmínka je volitelný parametr pravidla a pokud není definována, je implicitně považována za splněnou. I pro vyhodnocení podmínky se používají parametry předchůdce. Jak výrazy v podmínce tak výrazy v následníkovi jsou složeny z různých aritmetických znamének, volání funkcí, konstant a formálních parametrů předchůdce.

Parametrické 0L-systémy [12] jsou definovány jako uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde

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

Ukázkou, jak je díky parametrům škála vytvářených slov rozšířena, je následující L-systém. Pomocí parametrů a různého nastavení konstant lze pomocí jediného pravidla vytvářet tvarově velice rozmanité struktury. Obrázek 2.6 zobrazuje čtyři příklady nastavení L-systému dle [14]. Všechny tyto stromové struktury byly vytvořeny pomocí následujícího jednoduchého L-systému:

$$\begin{aligned} \omega &: A(100, w_0) \\ p_1 &: A(s, w) : s \geq \min \rightarrow !w F(s)[+(\alpha_1)/(\varphi_1)A(s \star r_1, (w \star q)^e)][+(\alpha_2)/(\varphi_2)A(s \star r_2, (w \star (1-q))^e)] \end{aligned}$$



Obrázek 2.6: Stromové struktury vytvořené pomocí parametrického L-systému.

2.6 Stochastické L-systémy

Všechna slova generovaná stejným deterministickým L-systémem jsou identická. Tato předvídatelnost je samozřejmě jednou z velkých výhod modelování pomocí L-systémů. Díky ní

2.6. STOCHASTICKÉ L-SYSTÉMY

Obrázek	r_1	r_2	α_1	α_2	φ_1	φ_2	w_0	q	e	min	n
a	.65	.71	27	-68	0	0	20	.53	.50	1.7	12
b	.92	.37	0	60	180	0	2	.50	.00	0.5	15
c	.80	.80	30	-30	137	137	30	.50	.50	0.0	10
d	.55	.95	-5	30	137	137	5	.40	.00	5.0	12

Tabulka 2.1: Hodnoty konstant pro vytvoření stromů z obrázku 2.6.

může uživatel přímo pozorovat dopady provedených změn. Tento determinismus má však také za následek, že modely zcela neodpovídají svým biologickým protějškům. V přírodě se totiž jedinci určitého rostlinného druhu často mezi sebou liší. Mohou se lišit stavbou i tvarem a přitom stále sdílet určité vlastnosti dané svým druhem. Pro simulaci tohoto jevu je dobré vnést do L-systému určitý druh náhodnosti, jež zároveň zajistí, že základní vlastnosti daného druhu budou zachovány a výslední jedinci se budou lišit jen ve vybraných detailech. Pro tento účel byly zavedeny stochastické L-systémy, které umožňují do přepisovacího procesu zavést náhodnost velice dobře kontrolovatelným způsobem.



Obrázek 2.7: Příklady různých jedinců generovaných jedním stochastickým L-systémem.

Takovýto parametrický stochastický OL-systém lze definovat jako uspořádanou pětici $G_\pi = \langle V, \Sigma, \omega, P, \pi \rangle$. Definice abecedy V , formálních parametrů Σ , axiomu ω a množiny přepisovacích pravidel P je stejná jako u parametrických L-systémů. Navíc je zde definována funkce π , která každému pravidlu z množiny P přiřazuje aritmetický výraz $E(\Sigma)$. Tento výraz nabývá vždy nezáporných hodnot a jeho výsledek se označuje jako *pravděpodobnostní faktor* přepisovacího pravidla. Pokud je při přepisovacím procesu více pravidel, které odpovídají určitému modulu, použije se pro výběr pravidla vzorec, který na základě pravděpo-

2.6. STOCHASTICKÉ L-SYSTÉMY

dobnostního faktoru určí pravděpodobnost použití pravidla. Tento faktor nenabývá obecně konstantních hodnot pro celý přepisovací proces, ale vyhodnocuje se, podobně jako podmínka a následník, na základě formálních parametrů předchůdce při každém použití pravidla. Pokud označíme $\hat{P} \subseteq P$ jako množinu všech odpovídajících pravidel jednomu modulu, pak pravděpodobnost $prob(p_k)$ jednoho pravidla $p_k \in \hat{P}$ odpovídá výsledku rovnice 2.6.1.

$$prob(p_k) = \frac{\pi(p_k)}{\sum_{p_i \in \hat{P}} \pi(p_i)}$$

Rovnice 2.6.1: Výpočet pravděpodobnosti výběru pravidla p_k

Kapitola 3

Modelování rostlinných organismů

Efektivní a reálné zobrazování rostlin je v oblasti počítačové grafiky velice aktuální a diskutované téma. Hlavním důvodem je, že se jedná o zcela běžně zobrazované objekty v různých aplikacích, scénách a simulacích. Jejich generování a zobrazení však na rozdíl od jiných často zobrazovaných modelů není tak jednoduché. Většina modelů je totiž tvořena relativně velkými plochami a lze je vytvořit malým množstvím polygonů. Rostlinné organismy jsou však tvořeny složitými strukturami malých objektů, které jsou navíc řízené určitými přírodními zákonitostmi. Jejich modely jsou tedy díky velkému množství polygonů složitější pro vykreslování. Kvůli složité vnitřní struktuře je navíc komplikované i jejich generování.

Tato kapitola se zabývá samotnou problematikou vytváření modelů rostlin. Jsou zde vysvětleny metody, jež byly v této práci použity pro návrh a modelování rostlinných organismů při zachování některých nejdůležitějších přírodních pravidel.

3.1 Interpretace želví grafikou

Protože byly L-systémy navrženy jako matematický model bez geometrické interpretace, je potřeba pro modelování rostlin použít některý z přístupů, jež pro tento účel byly buď vyvinuty nebo byly převzaty. Sám Lindenmayer publikoval v roce 1974 řešení jež nahrazovalo moduly řetězců grafickými obrazci. Šlo však hlavně o topologii větvení rostlin a detaily jako délky nebo úhly natočení segmentů byly do modelu dodávány dodatečně. Během sedmdesátých a osmdesátých let vzniklo ještě mnoho jiných interpretací L-systémů, jež například ukázaly, že L-systému jsou velmi platným nástrojem pro tvorbu fraktálů. V roce 1986 přišel Przemysław Prusinkiewicz s myšlenkou interpretovat L-systémy jako pohyb želvy známé z programovacího jazyka LOGO [1]. Tato želva funguje jako kurzor, který přijímá různé příkazy týkající se jeho pohybu. Na základě těchto příkazů pak tento kurzor kreslí svým pohybem jednu spojitou čáru. Použití želví grafiky značně rozšiřuje geometrické možnosti L-systémů a je ideální pro tvorbu rostlinných struktur.

3.1.1 Planární želví grafika

Jelikož původ želví grafiky spočívá v pohybu kurzoru po obrazovce, její původní koncept zahrnoval pohyb ve dvojrozměrném prostředí. Želva je definovaná jako trojice parametrů (x, y, α) , kde x a y udávají kartézké souřadnice želvy a úhel α směr, kterým želva míří. Jde o tzv. čelo želvy. Pokud zavedeme navíc hodnotu pro délku kroku a výchozí přírůstek

3.1. INTERPRETACE ŽELVÍ GRAFIKOU

úhlu, o který se želva bude otáčet, můžeme vytvořit jednoduchá pravidla pro pohyb želvy v rovině. Jednotlivé znaky pak reprezentují daný modul a také akci, která je při načtení tohoto modulu provedena.

F Želva provede krok vpřed o předem definované délce na pozici (x', y') . Mezi body (x, y) a (x', y') je vykreslena úsečka.

- + Přikáže želvě otočit se o předem definovaný přírůstek úhlu doleva. Úhel želvy se zvětší o tuto hodnotu.
- Přikáže želvě otočit se o předem definovaný přírůstek úhlu doprava. Úhel želvy se zmenší o tuto hodnotu.

Vykreslování pomocí želví grafiky probíhá tak, že želva s počátečními parametry (x_0, y_0, α_0) interpretuje jednotlivé znaky řetězce postupně zleva doprava.

3.1.2 Interpretace závorkových L-systémů

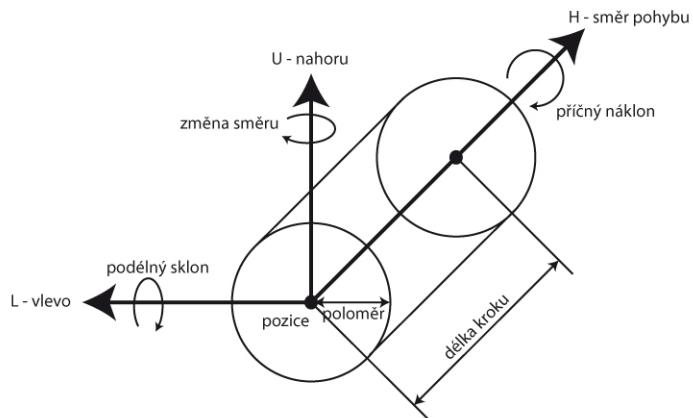
Jak již bylo výše zmíněno, pro simulaci větvících se struktur rostlin je nutné rozšíření L-systémů o závorky. Každý podřetězec uzavřený závorkami zleva i zprava představuje v topologii rostliny jednu větev. Při použití želví grafiky je tak nutné v bodech větvení rozdělit souvislou čáru, jež želva kreslí, na minimálně dvě nové cesty. Pro tyto účely se užívá zásobníku. Pokud želva narazí při procházení slova na znak [, uloží se spolu se svými parametry na zásobník a její kopie pokračuje v interpretaci řetězce za závorkou. Naopak pokud želva narazí na znak], je odstraněna a v interpretaci pokračuje želva z vrcholu zásobníku.

3.1.3 Rozšíření pro interpretaci L-systémů v 3D

Jednou z hlavních výhod želví grafiky je její jednoduchá rozšiřitelnost pro použití ve třech rozměrech. Parametry želvy však musejí být rozšířeny. Pro pozici želvy je zaveden vektor \vec{P} . Orientaci určují tři jednotkové vektory \vec{H} , \vec{L} a \vec{U} takové, že splňují rovnici $\vec{H} \times \vec{L} = \vec{U}$. Vektor \vec{H} , podobně jako u rovinné želvy, udává kam želva směřuje. Vektor \vec{U} směřuje směrem nahoru a vektor \vec{L} směřuje vlevo od želvy.

Sada příkazů je v 3D také značně rozšířena. Pro změnu orientace želva interpretuje příkazy, jež želvu rotují okolo jednotlivých os. Pro tyto rotace jsou běžně používány termíny z letectví: kurs, podélní sklon a příčný náklon. Kompletní sada příkazů, kterou interpretuje i zásuvný modul, jež byl vytvořen v rámci této práce, je i spolu s popisem interpretovaných funkcí uvedena v příloze A.

3.2. SUBAPIKÁLNÍ L-SYSTÉMY



Obrázek 3.1: Polohové a pohybové parametry želvy v 3D prostoru.

3.2 Subapikální L-systémy

Závorkové L-systémy umožňují vytvářet velice širokou škálu řetězců. Ne všechny však odpovídají rostlinné stavbě a pro generování rostlin se tak nehodí. Zavádějí se proto jistá omezení a pravidla, obvykle nazýváné jako vzory větvení, jež nám zaručí respektování některých přírodních vlastností. Výsledkem pak jsou L-systémy, které pravdivěji odpovídají reálným rostlinám. Jednou z těchto podmnožin jsou i subapikální L-systémy. Poprvé byly představeny Alicí Kelemenovou v roce 1986 [7]. Myšlenkou těchto L-systémů je, že k větvení dochází pouze u vrcholů již existujících větví. Tato vlastnost vychází ze základních výsledků při pozorování rostoucích rostlin. Nové rostlinné orgány, jako je stonk, větve, listy nebo květy mohou být vytvořeny pouze z apikálního meristému. Tato tkáň obsahuje aktivně se dělící buňky a nachází se v oblasti vrcholů větví.

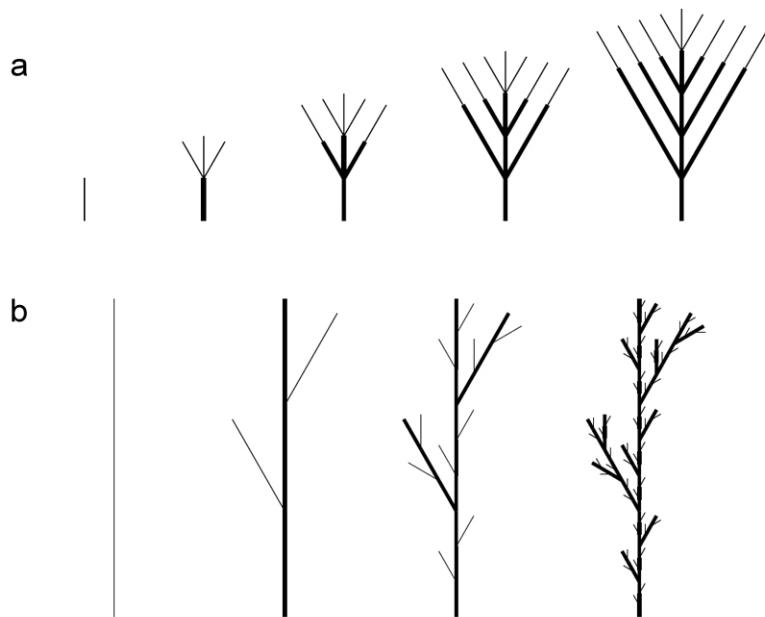
3.3 Vzory větvených struktur

Jedním z nejvýznamnějších parametrů rostlin je způsob větvení. Má totiž velice zásadní vliv na vzhled. Zařazením rostliny do některého z typů větvení lze také někdy odvodit typ L-systému, který bude nutné pro vygenerování modelu použít. Tato kapitola obsahuje základní vzory větvení rostlin, jež se v přírodě vyskytují.

3.3.1 Monopodiální vzor

Monopodiální struktury se vyznačují jedním hlavním souvislým stonkem. Z něj pak vychází mnoho vedlejších větví. Jelikož hlavní kmen je vždy nejstarší větví, jeho tloušťka je největší. Vedlejší větve se pak vyvíjejí podle stejného vzoru. Tento vzor lze dále rozdělit podle distribuce vedlejších větví na hlavním stonku.

3.3. VZORY VĚTVENÝCH STRUKTUR



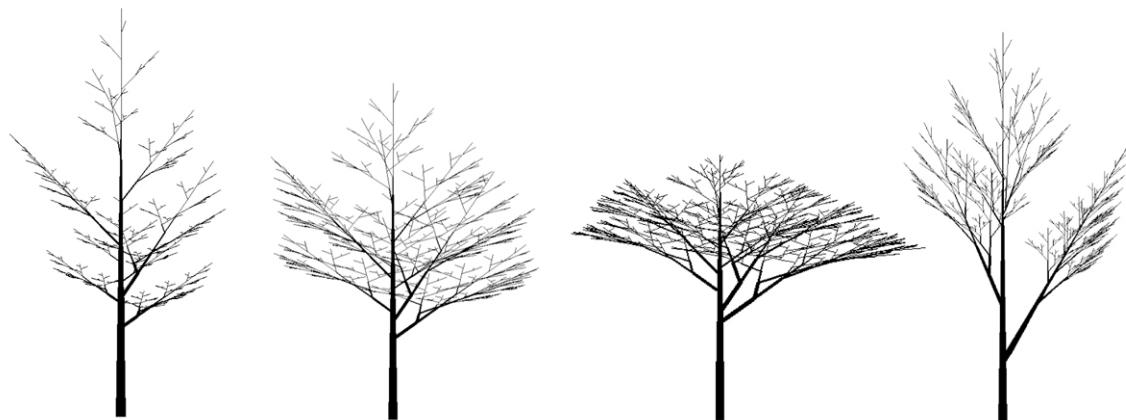
Obrázek 3.2: První řada (a) zobrazuje vývoj struktury, jež splňuje podmínu subapikálního větvení. Druhá větvená struktura (b) ji nesplňuje[13].

- *Bazitonické struktury* mají větve u vrcholu rostliny méně vyvinuté, než větve blíže základny rostliny. Lze je vytvořit i za pomocí jednoduchých D0L-systémů. Pokud je totiž při přepisování dodržena subapikální podmínka a je postupováno tak, že moduly vrcholu jsou přepisovány určitým počtem meziuzlů zakončených opět vrcholy, je zřejmé, že větve blíže kořenů budou rozvinutější. Je na nich totiž provedeno větší množství iterací. V přírodě se jedná o jednodušší organismy. Typickými zástupci jsou například jehličnaný nebo cykasy.
- *Mezotonický a akrotonický typ větvení* lze v přírodě také nalézt. U takto stavěných rostlin jsou nejrozvinutější větve uprostřed, respektive u vrcholu stonku. Pro větvení dále od počátku větve je však zapotřebí zavést do L-systému signály nebo růstový potenciál. Tyto mechanismy však nelze vytvořit pomocí D0L-systémů. Propagaci signálu lze provést pomocí kontextu u kontextových L-systémů. Zvyšování růstového potenciálu lze provést naopak pomocí parametrických L-systémů.

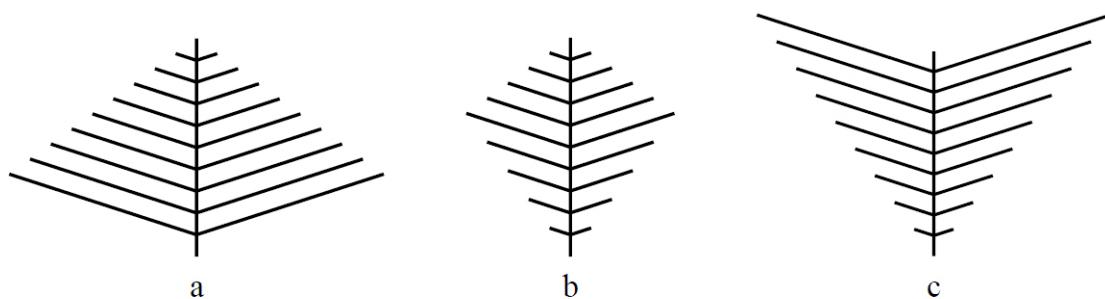
3.3.2 Sympodiální vzor

Jedná se o vývojově pokročilejší typ větvení. Hlavní stonek je nesouvislý a jeho růst je brzy potlačen. Vedlejších větví je zde menší počet. Jsou však o to více rozvinutější. Většina listnatých stromů tvoří právě sympodiální struktury. Modely s touto strukturou lze vytvořit i za

3.4. VLASTNOSTI VĚTVENÝCH STRUKTUR ROSTLINNÝCH ORGANISMŮ



Obrázek 3.3: Ukázky monopodiálního větvení stromů [12].



Obrázek 3.4: Bazitonické (a), mezotonické (b) a akrotonické (c) dělení [13].

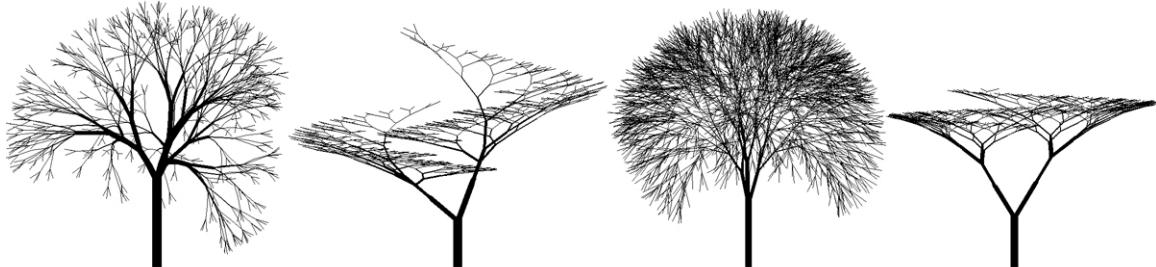
pomocí D0L-systémů. Docílíme však pouze jednoduše se délících a opakujících se topologií. Pro pokročilejší modely je možné použít i jiné typy L-systémů.

3.4 Vlastnosti větvených struktur rostlinných organismů

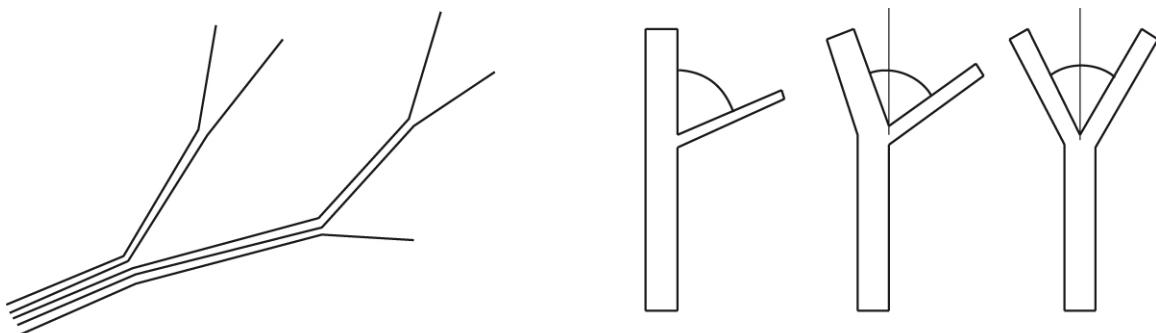
Ačkoli se větvení stromů i jiných rostlin může jevit náhodné, platí i pro ně určitá pravidla. Již Leonardo da Vinci zkoumal některé zákonitosti větvení stromů a stanovil postulát. Ten říká, že obsah průřezu větve před větvením je shodný se součtem všech obsahů průřezů větví za větvením. Později se ukázalo, že tento odhad byl velice přesný a je vhodné jej při návrhu L-systému dodržovat.

V roce 1994 tento postulát rozšířil Matthew Holton [6]. Na rozdíl od da Vinciho však již znal vnitřní strukturu větví a také princip cévních svazků u cévnatých rostlin. Cévní svazky jsou součástí transportního systému rostlin a rozvádějí živiny z kořenů rostliny do jejího

3.4. VLASTNOSTI VĚTVENÝCH STRUKTUR ROSTLINNÝCH ORGANISMŮ



Obrázek 3.5: Ukázky sympodiálního větvení stromů [12].



Obrázek 3.6: Distribuce cév ve větvích a závislost množství cév na úhlu dělení.

těla. Při větvení dochází k rozdělení svazku na dva či více samostaných svazků. Toto rozdělení právě souvisí s da Vinciho postulátem. Kvůli menšímu množství živin, mají rozvětvené větve adekvátně menší obsah průřezu. Holton však navíc při modelování svých rostlin použil zajímavou spojitost množství cévních svazků a úhlu dělení větví. Cévní svazky totiž plní také zpevňovací funkci. Zavedl vztah, podle něhož lze množství cév, a tedy i tloušťku větve při dělení, určit podle úhlu odchylky od původní osy. Čím je tento úhel větší, tím menší je počet cév a průměr větve.

Kapitola 4

Zásuvný modul LSystem

Stěžejní částí této diplomové práce je návrh a implementace zásuvného modulu pro systém Vrecko. Tento modul umožňuje simulaci rostlin, stromů a jiných modelů pomocí různých typů L-systémů a jejich následnou grafickou interpretaci a zobrazení v systému Vrecko. Návrh a analýza byla provedena s velkým důrazem na modularitu a tedy možnost pozdějšího rozšíření o nové funkce.

Logicky lze zásuvný modul LSystem rozdělit na tři části. První celek tvoří syntaktický analyzátor souborů L-systémů a generátor slov. Syntaktický analyzátor zajišťuje správné zpracování L-systémů, jejich náhrání ze souboru a načtení všech potřebných parametrů. Následně pak zpracuje načtená data, která generátor použije pro provedení jednotlivých iteračních kroků L-systémů. Slova i ostatní data jsou během iteračního procesu uložena do datových struktur optimalizovaných pro daný účel. Jelikož existuje několik typů L-systémů, je implementována funkcionality, která pro přepisování vybere algoritmus, který je svými schopnostmi i časovou náročností nevhodnější pro zpracovávaný L-systém. Tvorba L-systémů se složitější strukturou se většinou neobejde bez hierarchického rozdělení problému. Zásuvný modul proto umožňuje rozdělit model na několik úrovní a do hlavního L-systému vkládat podsystémy. V praxi to pak například u generování stromů znamená, že je možné vytvořit rozdílné L-systémy pro listy, květy a plody, které lze následně vložit do L-systému kostry stromu.

Druhý celek se zabývá grafickou interpretací vygenerovaného řetězce. S dříve zmiňovaným celkem komunikuje pouze na úrovni předání vygenerovaného slova, poskytnutí detailních parametrů pro nastavení interpretu a také mechanismu pro zpracování dotazů. Interpret v současné podobě používá pro generování geometrie modelů výhradně želví grafiku. Tento modul však obsahuje rovněž interpret, jenž k vytváření geometrie neslouží. Jedná se o interpret pro zpracování dotazů během generování iterací L-systémů. Je tedy určen pro rychlé zjištění polohy a orientace želvy. Pro zpracování slov obsahujících závorky má interpret implementován zásobník. Uživatel má také možnost volit z několika typů želv, které mu dovolují vykreslovat geometrii rozdílnými přístupy. Poslední ze tří celků plní funkci řídícího modulu a je umístněn nad oběma předcházejícími celky, jež spravuje. Zároveň sprostředkovává komunikaci jak se systémem Vrecko tak mezi moduly navzájem. Jedná se především o načtení základních parametrů, které oba moduly potřebují pro svou inicializaci.

4.1 Vývoj zásuvného modulu

Vývoj probíhal na platformě Windows v prostředí Visual Studio 2005 v jazyce C++. Veškeré třídy náleží do jmenného prostoru AP_LSystem. Stejně jako celý systém Vrecko je i modul LSystem postaven na OpenSceneGraphu. Zejména se jedná o část zajišťující interpretaci L-systémů a generování geometrie. Mimo to byly pro vývoj použity některé z knihoven Boost. Knihovna boost::program_options umožňuje jednoduché a intuitivní ukládání a zpracovávání nastavení a konfiguračních souborů. Knihovna boost::lexical_cast jednoduše převádí a přetypovává řetězce znaků. Z balíku Boost je také použit inteligentní ukazatel boost::shared_ptr. O zpracování XML souborů se stará knihovna Xerces-C.

4.2 OpenSceneGraph

Základním kamenem pro modul LSystem i pro systém Vrecko je OpenSceneGraph API. Jedná se o objektově orientované rozhraní nad OpenGL. Nízkoúrovňová funkcionality je zde převedena na objekty v grafech scén. Zároveň jsou možnosti OpenSceneGraphu oproti klasickému OpenGL značně rozšířené a je přidáno mnoho funkctionalit, jež zjednoduší a urychlují vývoj.

4.3 Vrecko

Jak již bylo zmíněno, je modul LSystem určen pro systém Vrecko. Jedná se o prostředí pro tvorbu virtuální reality vyvíjené v rámci HCI laboratoře na Fakultě informatiky Masarykovy univerzity v Brně. Slouží primárně pro vytváření scén, jež nějakým způsobem demonstруjí a zkoumají možnosti interakce člověka s počítačem. Vychází z principu OpenSceneGraph a scény jsou tedy obdobně definovány jako grafy. Scény se v tomto prostředí dělí na objekty, které mají určité vlastnosti a dovednosti. Díky nim lze objekty ovládat a měnit jejich tvar i vlastnosti. Zároveň lze k tomuto systému připojit řadu různých zařízení pro interakci uživatele s obsahem scény.

Kapitola 5

Generování slov pro modely rostlin

Tato kapitola se zabývá implementací syntaktického analyzátoru a následným generováním slov pomocí L-systémů. Při implementaci této části byl kladen velký důraz na rychlosť a také na modularitu. Rychlosť je zde důležitá zejména při iteračním procesu, neboť výsledné slovo může mít délku až v řádu statisíců modulů. Je tedy třeba zajistit vhodné struktury pro ukládání dat a vybrat vhodný algoritmus pro rychlý přepis pravidel. Modularita a rozšiřitelnost zde s rychlosťí úzce souvisí. Jednotlivé typy L-systémy totiž vyžadují rozdílné přístupy při přepisování pravidel a tak je důležité vytvořit více algoritmů, jež budou různé typy jednodušších i komplexnějších L-systémů zpracovávat. Zároveň je však třeba zaručit, aby jednodušší L-systémy nebyly zpracovány příliš složitými algoritmy, ale aby byly vybrány jednodušší a hlavně optimalizovanější postupy pro daný problém. Modularita je zde důležitá i pro syntaktický analyzátor, který tak zvládne zpracovat více formátů L-systémů.

Celou tuto funkcionality a komunikaci s okolím zajišťuje třída `LSystemGenerator`. Tato třída je potomkem třídy `AbstractGenerator`, která obsahuje rozhraní pro obsluhu této třídy. Instance třídy `LSystemGenerator` obsahuje odkaz na hlavní L-systém třídy `AbstractLSystem`, který je poté použit pro generování slova. Hlavním úkolem třídy `LSystemGenerator` je vybrat správný syntaktický analyzátor pro nahrání souboru s L-systémem a na základě získaných parametrů vybrat a vytvořit instanci třídy dědící z `AbstractLSystem`, jež bude pro následný iterační proces nevhodnější. Nahráním souborů a jejich strukturou se zabývá následující kapitola.

5.1 Soubory L-systémů

Vstupním bodem pro každý L-systém je jeho definice uložená v jednom z podporovaných souborových formátů. Spolu s definicí může být součástí celá řada parametrů, které mohou ovlivnit jak proces přepisování pravidel tak i způsob následné geometrické interpretace. Zásuvný modul nyní podporuje dva typy souborů a lze jej rozšířit o podporu dalších formátů. Jedná se o textový formát LS, jenž v obdobné formě používá ve svých projektech výzkumná skupina Biologického Modelování a Vizualizace kolem prof. Przemysława Prusinkiewicze z University of Calgary. Tento formát byl zvolen z důvodu vedoucí úlohy této skupiny v oboru L-systémů. Druhým formátem je soubor napsaný ve značkovacím jazyce XML, jenž byl zvolen pro svou jednoduchou rozšiřitelnost a přehlednost.

Struktura je v obou případech hierarchická a popis jejich entit je následující:

5.1. SOUBORY L-SYSTÉMŮ

- **Unikátní identifikátor L-systému** je nutné přiřadit každému L-systému. Ten se pak používá hlavně při vkládání podřízených L-systémů nebo pro získávání parametrů o daném L-systému.
- **Typ L-systému** je důležitým a nutným parametrem L-systému. Určuje například, zda je L-systém deterministický či bezkontextový. Na jeho správném nastavení pak závisí výběr správného algoritmu pro zpracování samotných pravidel a pro následný iterační proces.
- **Parametry** nastavené přímo v souboru L-systému jsou specifické pro konkrétní L-systém. Pokud některé z parametrů nejsou nastaveny zde, použijí se pro generování a interpretaci slova globální parametry získané z konfiguračního souboru. Kompletní popis jednotlivých parametrů se nachází v příloze B.
- **Podřízené L-systémy** lze vkládát do každého L-systému. Lze je vložit přidáním cesty k souboru L-systému. Nikdy však není vhodné tvořit v grafu hierarchie L-systémů kružnice. Podrobněji se touto tématikou zabývá kapitola 5.7.
- **Axiom** je počáteční řetězec symbolů. Jde tedy o slovo v nulté iteraci L-systému.
- **Přepisovací pravidla** tvoří množinu pravidel, která budou použita při iteračním procesu. Na pořadí není brán zřetel, protože se při každém iteračním kroku aplikují všechna pravidla najednou. Z tohoto důvodu je důležité, aby byla jednotlivá pravidla jednoznačná a neexistovala dvě pravidla pro stejný modul předchůdce. Jedinou výjimkou jsou stochastické systémy. Struktura pravidel je závislá na jejich typu a podrobněji je popsána v odpovídajících podkapitolách kapitoly 5.5.
- **Homomorfismy** jsou zvláštním typem přepisovacích pravidel, která se nezpracovávají během iteračního procesu. K přepisu těchto pravidel dochází vždy až po dokončení všech iterací.

Formáty se implementují jako potomci třídy `AbstractFile`. Ta poskytuje rozhraní pro získání potřebných dat ze souboru.

```
class AbstractFile
{
protected:
    unsigned m_Type;
    std::string m_Name, m_Axiom;
    std::vector<std::string> m_Rules, m_Homomorphisms, m_Subsystems;
    void substitute(std::map<std::string, std::string> &)
public:
    AbstractFile();
    virtual void open( std::string & ) = 0;
    std::vector<string> * getHomomorphisms();
    std::vector<string> * getRules();
    std::vector<string> * getSubsystems();
```

```

    std::string & getAxiom();
    unsigned getType();
    std::string & getName();
};

}

```

Třídy odvozené od třídy `AbstractFile` musí implementovat metodu `open()`, jež zařučí zpracování dat ze souboru a uloží je do jednotlivých atributů této třídy. Pomocná metoda `substitute()` nahrazuje řetězce a může být použita pro zpracování konstant nebo podřízených L-systémů. Ostatní metody slouží jen jako přístupové metody k atributům.

5.1.1 Formát LS

Jedná se o jednoduchý textový formát založený na příkazech podobných preprocesorovým příkazům jazyka C. Některé příkazy jsou párové a musí být ukončeny. Jako hodnota je brán řetězec následující po příkazu. Tento řetězec nemůže obsahovat žádné bílé znaky. Formát LS dovoluje také vkládat komentáře použitím dvojitého lomítka //.

```

#lsystem TernaryTree           // unikátní ID L-systému

#type 0L                      // typ L-systému
#define Pitch 65                // konstanta
#include data\ls\leaf.ls        // cesta k podřízenému L-systému

// parametry
#set Iteration=14
#set TurtleType=STRAIGHT_PIPE
#set DefaultAngle=30.0

// axiom
#axiom
F' (0.65)A
#endaxiom

// přepisovací pravidla
#rules
A:->! (0.577)' (0.87) [/ (90.74)B] [/ (-132.63)B]B
B:->^ (33.95) [^(Pitch) #{Leaf}] [#{Leaf}]Z[^ (Pitch) #{Leaf}] [#{Leaf}]ZA
#endrules

#endlsystem

```

5.1.2 Formát XML

V případě XML souboru jde o klasický XML dokument verze 1.0 s kódováním UTF-8. Veškeré použité elementy jsou párové a nejsou použity žádné atributy. `LSystem` je kořenová značka, která obsahuje všechny potřebné entity. V následujícím odstavci je příklad zápisu L-systému do XML souboru.

```

<?xml version="1.0"?>
<!-- kořenová značka -->
<LSystem>
    <!-- unikátní identifikátor L-systému -->
    <Name>TernaryTree</Name>
    <!-- typ L-systému -->
    <Types>
        <Type>0L</Type>
    </Types>
    <!-- konstanty -->
    <Constants>
        <LeafPitch>65.0</LeafPitch>
    </Constants>
    <!-- parametry -->
    <Parameters>
        <Iteration>14</Iteration>
        <TurtleType>STRAIGHT_PIPE</TurtleType>
        <DefaultAngle>30.0</DefaultAngle>
    </Parameters>
    <!-- subsystémy -->
    <Subsystems>
        <Subsystem>data\ls\leaf01.ls</Subsystem>
    </Subsystems>
    <!-- axiom -->
    <Axiom>F' (0.65) A</Axiom>
    <!-- přepisovací pravidla -->
    <Rules>
        <Rule>A:->! (0.577)' (0.87) [/ (90.74) B] [/ (-132.63) B] B</Rule>
        <Rule>B:->^ (33.95) [^(LeafPitch) # {Leaf01}] [# {Leaf01}] Z
            [^(LeafPitch) # {Leaf01}] [# {Leaf01}] ZA</Rule>
    </Rules>
    <!-- homomorfismy -->
    <Homomorphisms>
        <Homomorphism>A:->Z</Homomorphism>
    </Homomorphisms>
</LSystem>

```

Některé symboly pravidel jsou bohužel zároveň řídícími symboly XML, a tak je zapotřebí nahradit jejich odpovídajícími XML entitami.

5.2 Řetězce modulů

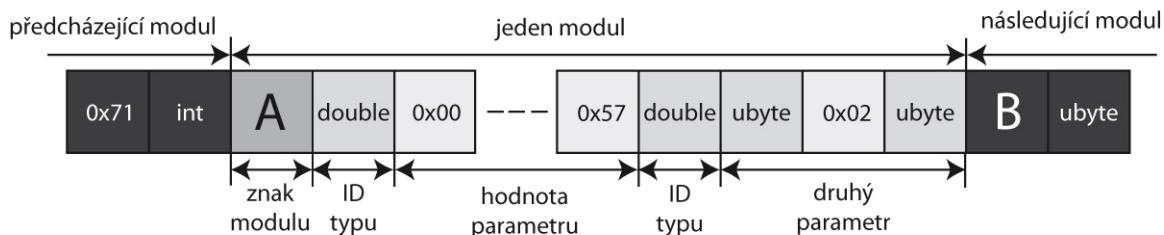
Při generování slova L-systémem dochází k vytváření velmi dlouhých řetězců s nestejnoro-dými daty. Moduly, ze kterých se řetězce tvoří, totiž kromě znaku pro identifikaci modulu obsahují také parametry různých datových formátů. Zpracování takovýchto dat musí být navíc dostatečně rychlé. Pro tento účel byla vytvořena na míru přizpůsobená třída `LongString`. Data jsou zde uložena ve formátu, který umožňuje rychlou manipulaci s daty, především

5.2. ŘETĚZCE MODULŮ

pak připojení nového řetězce na konec stávajícího. Při iteračním procesu je zapotřebí provádět tuto operaci rychle. S každou iterací je zpracováno původní slovo a podle něj se vytváří vytváří, na základě přepisovacích pravidel, nové slovo další iterace. Provádí to tak, že prohlíží jednotlivé moduly původního slova zleva doprava a hledá shodu s některým z předchůdců v přepisovacích pravidlech. V případě nalezení pravidla je následník přidán na konec nového slova. V případě, že předchůdce mezi pravidly nalezen není, připojí se na konec kopie původního modulu. Použije se tedy pravidlo identity. Implementací samotného přepisovacího mechanismu se zabývá kapitola 5.4.3

5.2.1 Vnitřní struktura řetězce

Datová struktura `LongString` je v jádru tvořena bajtovým polem. Toto pole se nezvětšuje při každém rozšíření řetězce, ale vždy jen po dosažení určité velikosti. Dochází tak ke znatelné úspoře času při alokaci paměti, která se provádí podstatně méně častěji. Každý modul je zde uložen jako znak identifikující daný modul a případně jeho parametry. Parametry mohou být typu `int`, `unsigned char` nebo typu `double`. Implementace dalších typů je možná a jednoduchá. Pro současné použití však nebyly jiné typy zapotřebí. Obrázek 5.1 ukazuje, jakým způsobem je struktura navržena.



Obrázek 5.1: Vnitřní uspořádání dat v řetězci typu `LongString`

Jedno políčko odpovídá jednomu bajtu. Každý modul obsahuje jeden znak, jenž jej identifikuje. Dále pak může obsahovat libovolný počet parametrů. Každý parametr je identifikován jedním bajtem zleva a jedním zprava. Při použití této datové struktury tedy není nutné provádět jakékoli převody mezi typy, jejichž hodnoty jsou tak rychle dostupné. Pro ilustraci byly na obrázku použity u jednoho modulu dva různé datové typy.

5.2.2 Inicializace

Pro vytvoření instance slouží jediný konstruktor. Parametrem je velikost řetězce. Implicitní hodnota tohoto parametru je 1 048 576 bajtů. Právě hodnota tohoto parametru je také použita jako minimální přírůstek alokované paměti, pokud se stávající paměť naplní. Plnění daty lze provádět buď připojováním řetězců, jež je popsáno v následující kapitole, nebo konverzí klasického zápisu řetězce typu `std::string` do podoby řetězce `LongString`.

K tomu slouží metoda `convertFromString()`. Této možnosti lze využít při načítání pravidel získaných ze souboru.

Identifikující znaky modulů jsou v nezměněné podobě uloženy i v řetězci `LongString`. Hodnoty parametrů jsou však konvertovány do odpovídajícího typu. Pro rozlišení typů se používají různé závorky. Zatímco kulaté závorky indikují typ `double`, složené závorky indikují typ `integer`. Konvertované hodnoty jsou v řetězci z obou stran obaleny jednobajtovým identifikátorem. Hodnota tohoto identifikátoru odpovídá pořadí ve výčtovém typu `ParameterType`.

5.2.3 Připojování řetězců

Jak již bylo výše zmíněno, je tato struktura optimalizována pro opakování zvětšování přidáváním řetězců na její konec. Při každém připojení se provádí kontrola, zda je alokované místo dostatečné pro připojení dalších dat. Pokud ne, volá se automaticky metoda `resize()`, jež alokuje novou paměť a zvětší tento řetězec o délku nastavenou při vytváření instance.

Pro připojení dat je k dispozici několik metod. Jedná se o různé formy metody `append()`. Ta poskytuje díky šabloně možnost uložit libovolný, ve výčtu `ParameterType` definovaný, datový typ. Kromě toho je tato metoda přetížena kvůli specifickým způsobům připojení některých dat. Lze tak připojit jiný řetězec typu `LongString` nebo pole bajtů ve správném formátu.

5.2.4 Přístup k datům řetězce

Přistupovat lze k datům jedním ze čtyř způsobů.

- Operátor `[]` přímo poskytuje hodnotu bajtu na určité pozici řetězce.
- Díky metodě `getData()` lze získat blok dat nebo celý řetězec bajtů. Hodí se především při provádění substitucí.
- Nejpoužívanější možností je metoda `getSymbol()`, díky níž lze získat blok dat odpovídající jednomu modulu, neboli symbolu, na dané pozici.
- Metody šablony `getParameters()` se používají pro zpracování parametrických L-systémů. Vrací pole parametrů libovolného typu a jejich počet. Tyto parametry jsou získány z pozice za identifikujícím znakem modulu.
- Metody `matchRight()` a `matchLeft()` slouží především k ověřování kontextu. Pro svou funkci využívají metodu `peekSymbol()` pro nahlednutí na levý nebo pravý nejbližší znak. Jelikož řetězce obsahují závorky, jsou implementovány i pomocné metody `findMatchingRightBracket()` a `findMatchingLeftBracket()` pro nalezení odpovídajícího protějšku k libovolné závorce. Podrobněji je celá funkcionalita nalezení kontextu popsána v kapitole 5.5.3.

5.3 Implementace přepisovacích pravidel

Potomci třídy `AbstractFile` zajišťují, aby byla pravidla správně načtena ze souboru. L-systému jsou předána jako klasický řetězec znaků. Ten však není vhodný pro samotný iterační proces. Proto se každé pravidlo zpracovává do struktury `Rule`. Různé typy L-systémů zpracovávají předané řetězce odlišně. Podrobněji se tomuto rozdělení věnuje kapitola 5.5. V této části jsou uvedeny implementační podrobnosti struktury `Rule`, do nichž všechny L-systémy svá přepisovací pravidla před iteračním procesem ukládají.

Jednou ze zásadních věcí bylo zakomponovat do pravidel nějaký mechanismus pro vyhodnocování výrazů. V pravidlech se totiž mohou výrazy vyskytovat hned na několika místech.

- Každý modul na straně následníka může místo parametru obsahovat výraz, jež se vyhodnocuje až na základě parametrů předchůdce.
- U parametrických L-systémů se může vyskytnout podmínka přepisu, která se musí před každým přepisem vyhodnotit. Při jejím nesplnění je pravidlo zamítnuto.
- Stochastické L-systémy mívají nastaven pravděpodobnostní faktor, na jehož základě závisí pravděpodobnost vybrání daného pravidla pro přepis. Tento faktor se může rovněž vyhodnocovat z výrazu.

Pro zpracování výrazů byla vybrána knihovna `FunctionParser`. Jedná se o volně šířitelnou knihovnu, která poskytuje dostatečnou a navíc uživatelsky rozšířitelnou funkcionality. Nabízí i možnosti optimalizace. Pro každý výraz, jenž je v pravidlech nalezen, je vytvořena jedna instance třídy `FunctionParser`. Pro zpracování slouží metoda `Parse()` s parametry zpracovávaného výrazu a řetězce všech proměnných. Vyhodnocení se provádí metodou `Eval()`, která po předání pole s hodnotami proměnných vrací výsledek výrazu.

Jako příklad je zde uvedeno pravidlo stochastického parametrického 2L-systému, kterým se zabývala kapitola 5.5.3.

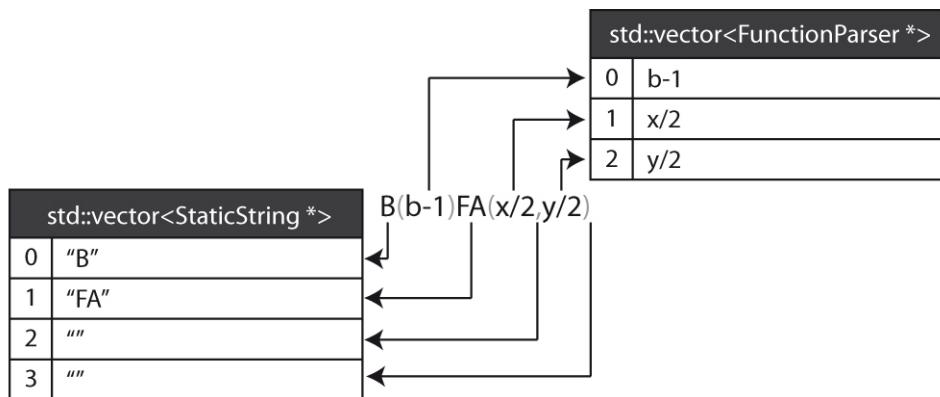
```
A(x,y) < B(z) > C(a,b,c) : (x>abs(a-c)) -> B(b-1) FA(x/2,y/2) : max(0,1-1/z*z)
```

Z tohoto zápisu je patrné, že pro všechny výrazy se používají proměnné předchůdce a kontextů. V tomto případě se jedná o `x`, `y`, `z`, `a`, `b` a `c`. Tyto znaky jsou uloženy ve struktuře `Rule` jako `m_Variables` a slouží k vytváření instancí třídy `FunctionParser`. Dále struktura obsahuje proměnné pro uložení znaků přechůdce, obou kontextů a ukazatele na instance pro vyhodnocení podmínky pravidla a pravděpodobnostního faktoru. Tyto proměnné pak slouží k výběru správného pravidla pro přepis modulu.

Jelikož se následník skládá z řetězců modulů a výrazů, je jeho zpracování složitější. Řetězce modulů jsou uloženy do instancí třídy `StaticString`. Jedná se o zjednodušenou variantu třídy `LongString`. Data jsou zde uložena ve stejné podobě. Rozdílem však je, že tento řetězec má nemennou velikost. Pro účel krátkých nemenných řetězců následníka je

5.4. ZPRACOVÁNÍ L-SYSTÉMŮ

tedy ideální, protože nezabírá tolik prostoru v paměti. Řetězce i výrazy se ukládají do kontejneru typu `std::vector`. Jsou zde uloženy tak, aby mohly být při přepisu tohoto pravidla střídavě z kontejneru vybírány. Ukázka zpracovaného pravidla z předchozího příkladu je na obrázku 5.2.



Obrázek 5.2: Rozdelení následníka na řetězce modulů a výrazy

Na obrázku 5.2 je vidět, že pokud po sobě následují dva výrazy, může být řetězec mezi nimi prázdný. Šedě jsou znázorněny znaky, které jsou zahozeny, neboť jsou pro další zpracování nepotřebné. Při zpracování i následném přepisu pravidla se vždy začíná a končí výběrem řetězce modulů. Velikost kontejneru s řetězci je tedy vždy právě o jednu větší než velikost kontejneru s výrazy. Je to z toho důvodu, že často existují pravidla, která obsahují v následníkovi pouze řetězec a žádný výraz. Pokud nějaký výraz obsahuje, vyskytuje se většinou někde uprostřed následníka. Proto je tento způsob zpracování optimální.

5.4 Zpracování L-systémů

L-systémy jsou implementovány jako potomci třídy `AbstractLSystem`. Abstraktní třída `AbstractLSystem` obsahuje několik metod sloužících jako rozhraní pro práci s L-systémy.

- Pro inicializaci slouží metoda `loadFromFile()`, která načte všechna potřebná nastavení. Jako zdroj nastavení slouží instance potomků abstraktní třídy `AbstractFile`.
- Pro provedení další iterace slouží metoda `nextIteration()`. Počet iterací lze ovlivnit i nastavením v konfiguračním souboru a v parametrech L-systému. Díky této metodě je však možné interaktivně provádět další iterace L-systému.
- Metoda `translate()` vrátí vygenerované slovo. Slovo je vždy uvnitř třídy L-systému uloženo ve své nejvyšší iteraci. Na slově, které tato metoda vrátí, jsou ještě předtím provedeny finální úpravy. Aplikují se pravidla homomorfismů a odkazy na podřízené L-systémy se nahradí slovy, jež tyto L-systémy stejným způsobem vygenerovaly.

5.4. ZPRACOVÁNÍ L-SYSTÉMŮ

Veškerá funkcionality, jež je společná pro všechny typy L-systémů je implementována ve společném rodiči `LSystem`. Celý proces se dá rozdělit do tří fází, jejichž popis je obsahem následujících kapitol.

5.4.1 Výběr nevhodnějšího L-systému

Zásuvný modul v současné podobě obsahuje tři různé třídy zpracovávající různé typy L-systémů. Liší se rychlostí a svými schopnostmi. Je tedy nutné, aby generátor vybral vhodný algoritmus. Stejně důležitý je i výběr u podřízených L-systémů. Každý soubor L-systému má nastaven svůj typ. Jedná se o výčet vlastností, které od třídy L-systému bude požadovat. Samotné třídy mají pak implementovanou statickou metodu `isCapable()`, na jejímž základě lze rozpoznat, zda daná třída požadavkům vyhovuje. Používá k tomu statický atribut `capabilities`, kde jsou schopnosti uloženy jako bitový součet. Jednotlivé schopnosti jsou definovány ve výčtu `LSystemCapabilities`.

Statická tovární metoda `AbstractGenerator::createLSystem()` usnadňuje vytváření nevhodnější instance. Na základě typu načteného souboru automaticky vytvoří a vrátí vhodnou instanci L-systému. Testování, zda daný typ L-systému vyhovuje, zde probíhá od nejrychlejšího typu algoritmu k nejpomalejšímu. První typ, který splňuje všechny požadavky, je vybrán a použit. Jelikož se jedná o tovární metodu, která nemůže zaručit uvolnění paměti, je zde jako návratová hodnota použit inteligentní ukazatel `boost::share_ptr`.

5.4.2 Inicializační fáze

Pravidla, která jsou předána jako řetězce, jsou v této fázi konvertována do struktury `Rule`. Ke konverzi slouží trojice metod. Metoda `setAxiom()` jednoduše přeloží řetězec znaků na vnitřní formát slova a nastaví jej jako počáteční slovo typu `LongString` pro iterační proces.

Nezbytnou součástí jsou metody pro přidání přepisovacích pravidel a homomorfismů. Struktura i funkce jsou u `addRule()` a `addHomomorphism()` dosti podobné. Způsob zpracování je v případě předchůdce shodný. Lišit se může zpracování následníka. Zpracování pravidel probíhá voláním jednotlivých funkcí struktury `Rule`, čímž se postupně prochází řetězec pravidla a instance struktury `Rule` je automaticky nastavována. Následující pseudokód ukazuje, jakým způsobem je zpracováno pravidlo parametrického stochastického 0L-systému.

```
přidejPřepisovacíPravidlo( řetězec zápisPravidla )
{
    Rule pravidlo;
    pravidlo.zpracujPředchůdce( zápisPravidla );
    pravidlo.zpracujPodmínu( zápisPravidla );

    // metoda zpracujStatickýŘetězecNásledníka() vrací:
    //   - true - pokud po zpracování následník pokračuje výrazem
    //   - false - pokud následník tímto statickým řetězcem končí
    for( pravidlo.zpracujStatickýŘetězecNásledníka( zápisPravidla ) )
```

```

    {
        pravidlo.zpracujVýrazVNásledníkovi( zápisPravidla );
    }
    pravidlo.zpracujPravděpodobnostníVýraz( zápisPravidla );
}

```

5.4.3 Přepisovací fáze

Jednoznačně nejdůležitější fází je samotný iterační proces. Probíhá v krocích po jednotlivých iteracích. Začíná vždy s počátečním slovem, axiomem, v němž jsou při první iteraci všechny moduly nahrazeny dle přepisovacích pravidel. Tento postup se opakuje při každé iteraci na nově vzniklých slovech. Celý proces přepisu je zjednodušeně zobrazen v následujícím pseudokódu.

```

LongString původníSlovo;
LongString novéSlovo;
for( všechny moduly v původníSlovo )
{
    najdi všechny pravidla, které mají aktuální modul jako předchůdce;
    if( existuje alespoň jedno takové)
    {
        if(vyber pravidlo)
        {
            novéSlovo.připoj( vygeneruj následníka modulu);
        }
        else
        {
            novéSlovo.připoj(použij pro daný modul pravidlo identity);
        }
    }
    else
    {
        novéSlovo.připoj(použij pro daný modul pravidlo identity);
    }
}

```

Prvotní prohledání pravidel vybere pouze ta, jež mají aktuální modul jako svého předchůdce. Takových pravidel může být hned několik. Mohou se lišit v kontextu, podmínce nebo mohou být vybrány na základě pravděpodobnosti. Výběr jednoho pravidla z této množiny provádí metoda `selectRule()`, v pseudokódu označená jako „vyber slovo“. Právě toto je funkce, v níž se jednotlivé L-systémy mohou nejvíce lišit a kde lze implementovat různé typy funkcionalit pro výběr pravidla. Detaily výběru pravidla u jednotlivých typů L-systémů rozebírá kapitola 5.5. Pokud žádné z pravidel nevyhovuje, použije se pravidlo identity, které pouze zkopíruje modul do nově vznikajícího slova. Vkládání do nového slova probíhá výhradně připojováním na jeho konec.

5.5 Rozdělení L-systémů

Jak již bylo zmíněno v kapitole 2 o typech větvených struktur, je nutné pro modelování pokročilejších rostlinných organismů použít složitější typy L-systémů. Zároveň však není kvůli optimalizaci vhodné použít stejné L-systémy pro generování jednoduchých struktur. Různé typy L-systémů mají navíc různé formy zápisů svých přepisovacích pravidel. V následující kapitole jsou popsány specifika implementace jednotlivých typů L-systémů a také formáty zápisů pravidel, jež tyto L-systémy zpracovávají. Jsou zde rozebrány pouze vlastnosti, které nebyly společné a tak byly v minulé kapitole 5.4 zmíněny jen okrajově.

5.5.1 D0L-systémy

Nejjednodušším L-systémem je tato varianta implementovaná třídou `D0LSys tem`. Jak již název napovídá, zpracovává pouze deterministické bezkontextové L-systémy. Pro řadu modelů je tento typ dostačejší. Navíc je díky své jednoduchosti nejrychlejší. Zápis pravidel je ukázán na následujícím vzorovém příkladu.

$A \rightarrow A + (10) FBF$

Tento typ nepodporuje zpracování výrazů. Každý následník se skládá pouze z jednoho řetězce typu `StaticString`. Iterační proces je v tomto případě vcelku jednoduchý. Vždy existuje maximálně jedno pravidlo, jehož předchůdce odpovídá právě zpracovávanému modulu původního řetězce. Toto pravidlo je také vždy vybráno a jeho následník je použit při přepisu.

Příklad stromu generovaného D0L-systémem je na obrázku 5.3 (a). Jedná se o ternární strom, jenž se při každém větvení rozdělí právě na tři větve. Z obrázku je velmi zjevná pravidelnost, která je hlavní nevýhodou D0L-systémů.

5.5.2 Parametrické stochastické bezkontextové L-systémy

Tento typ L-systému umožňuje zpracovat parametrické i stochastické L-systémy, čímž značně rozšiřuje škálu možností. Díky parametrym lze vytvořit i mezotonické a akrotonické struktury. Náhodnost pak dovoluje vytvářet jedince rostlinných organismů s rozdílnou náhodnou strukturou, které však zároveň mají společné rodové znaky. Implementace se nachází v třídě `ParStoch0LSys tem`. Tato třída zpracovává L-systémy ve tvaru, jak je popisuje kapitola 2.5. Příkladem mohou být následující pravidla zapsaná již v podobě, která je tímto L-systémem zpracovatelná.

$A(x, y) : (x > y) \rightarrow A(x-1, y) FB(y/2) F : (x * y)$

Jednotlivé části tohoto řetězce mají následující význam.

- $A(x, y)$ označuje předchůdce pravidla se dvěma parametry x a y . Tento údaj je povinný.

- $(x>y)$ je příklad podmínky. Pokud pravidlo podmínuje, může být podmínka nahrazena symbolem $*$, který značí, že je podmínka splněna vždy.
- Následník $A(x-1,y)FB(y/2)F$ je rozdelen a zpracován do struktury Rule.
- (x^*y) je pravděpodobnostní faktor. Pokud se jedná o deterministické pravidlo, lze tento výraz vynechat a ukončit pravidlo posledním znakem následníka. V takovém případě je však nutné, aby žádné z pravidel s tímto předchůdcem neobsahovalo pravděpodobnostní faktor.

Výběr konkrétního pravidla provádí přetížená virtuální metoda `selectRule()`, která na vstupu obdrží už jen pravidla se shodujícím se předchůdcem. Tato pravidla je nutné dále protřídit. Pokud se nejedná o stochastický L-systém, vybere se pravidlo, které splňuje svou podmínku. V případě stochastického L-systému je běžné, že podmínuje několik pravidel. Zde je zapotřebí vyhodnotit jejich pravděpodobnostní faktory a výsledky pak předat instanci pomocné třídy `RandomIndex`. Ta na základě těchto faktorů vybere náhodně jedno z pravidel, které je pak použito pro přepis.

Ukázkou vložení parametrického prvku do modelu jsou obrázky 5.3 (b) a (c). Jedná se o jednoduchý D0L-systém ternárního stromu, ve kterém je však pomocí jednoduchých parametrických pravidel dosaženo jevu, kdy listy rostou pouze na mladších větvích. Starší a tlustší větve jsou, dle subapikálních pravidel růstu, bez listů. Listy na původním D0L-systému lze vytvořit pouze tak, že jsou rovnoměrně a pravidelně rozprostřeny na všech větvích.

Obrázek 5.3 (b) demonstriuje použití pravděpodobnostního faktoru. Výrazy pro jeho výpočet byly zvoleny tak, že s rostoucí vzdáleností od kořene klesá pravděpodobnost dělení větví. Je toho dosaženo pomocí dvou pravidel, z nichž jedno obsahuje dělení na dvě větve a druhé pouze krok vřed. Při prvních krocích je pravděpodobnost rozdelení téměř stoprocentní. Kolem třetího kroku však prudce klesá a dále je naopak minimální. Podobných pravidel lze v přírodě nalézt celou řadu. Pomocí parametrických stochastických L-systémů je lze vcelku jednoduše nasimulovat.

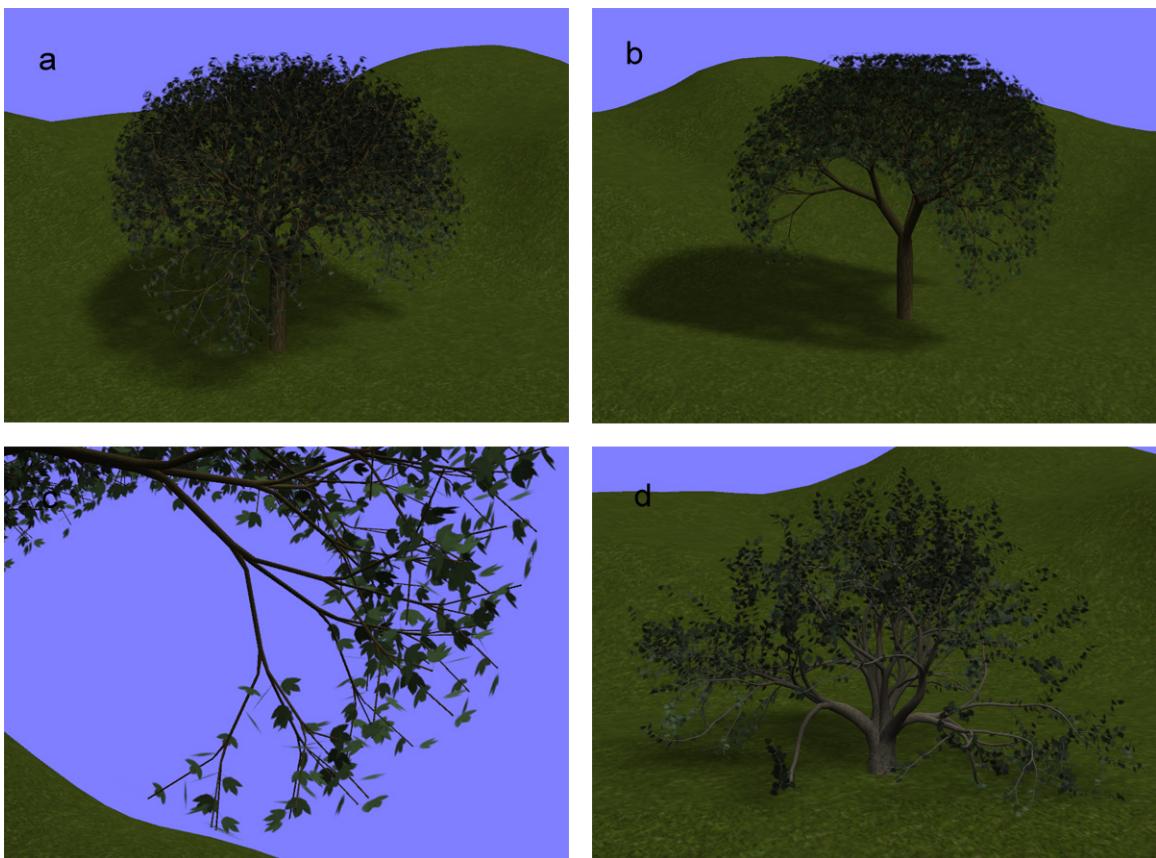
5.5.3 Parametrické stochastické kontextové L-systémy

Aby byl zásuvný modul schopen zpracovávat dotazy a signály, obsahuje také třídu pro zpracování kontextových L-systémů. Použití kontextu také znamená jiný zápis pravidel.

`A(x,y) < B(z) > C(a,b,c) : (z > abs(a-c)) -> B(b-1)FA(x/2,y/2) : max(0,1-1/z*z)`

Oproti bezkontextovým gramatikám se zde navíc zadává levý a pravý kontext. Jeho zadání lze vynechat, avšak znaménka $<$ a $>$ je nutné v řetězci pravidla zachovat. Zpracování podmínky i pravděpodobnostního faktoru je zde obdobné jako u bezkontextových L-systémů.

Nejdůležitějším rozšířením je schopnost ověřovat kontext předchůdce. Ověřování kontextu se řídí pravidly popsanými v kapitole o kontextových L-systémech 2.4. Pro tuto činnost byly implementovány metody `matchRight()` a `matchLeft()`. Jde o metody třídy



Obrázek 5.3: Bezkontextové L-systémy.

LongString a vrací hodnotu integer, která informuje zda bylo ověřování úspěšné. V případě úspěchu tyto metody vrací pozici úspěšně ověřeného modulu. Jinak vrací -1. Metody instance třídy LongString tedy dostanou informace, jaký kontext a kde jej ověřovat a poté samy vyhodnotí výsledek. Jak vyplývá z výše zmíněných pravidel, je potřeba u L-systémů se závorkami provádět někdy ověření na více místech zároveň. Jedná se například o situaci symbolu vedle závorky nebo pokud je sousední symbol v seznamu ignorovaných znaků. Ověřovací metody tak uvnitř obsahují rekurzivní volání sebe sama, aby všechny tyto situace byly korektně vyhodnoceny. Pokud dochází při ověřování k větvění, je většinou nutné naleznout v řetězci správná místa pro ověření kontextu tak, aby byla respektována skutečná topologie modelu. K tomuto účelu slouží metody findMatchingRightBracket () a findMatchingLeftBracket (), které k nalezené závorce najdou její protějšek. Tímto umožní ověřovacímu mechanismu pokračovat za závorkou. Pseudokód fungování metody matchRight () zjednodušeně ukazuje princip fungování této metody. U metody matchLeft () je princip obdobný.

```

int matchRight( kontext,    /*symbol pravého kontextu*/
                pozice,      /*odkud hledat*/
                ignorovat   /*seznam ignorovaných znaků*/ )
{
    if ( slovo[pozice] == '[' )
        return matchRight( kontext, pozice + 1, ignorovat ) ||
               matchRight( kontext, findMatchingRightBracket(), ignorovat );
    else if ((slovo[pozice] == '[') || (ignorovat obsahuje slovo[pozice]))
        return matchRight( kontext, pozice + 1, ignorovat );
    else if ( slovo[pozice] == kontext )
        return pozice;
    else
        return -1;
}

```

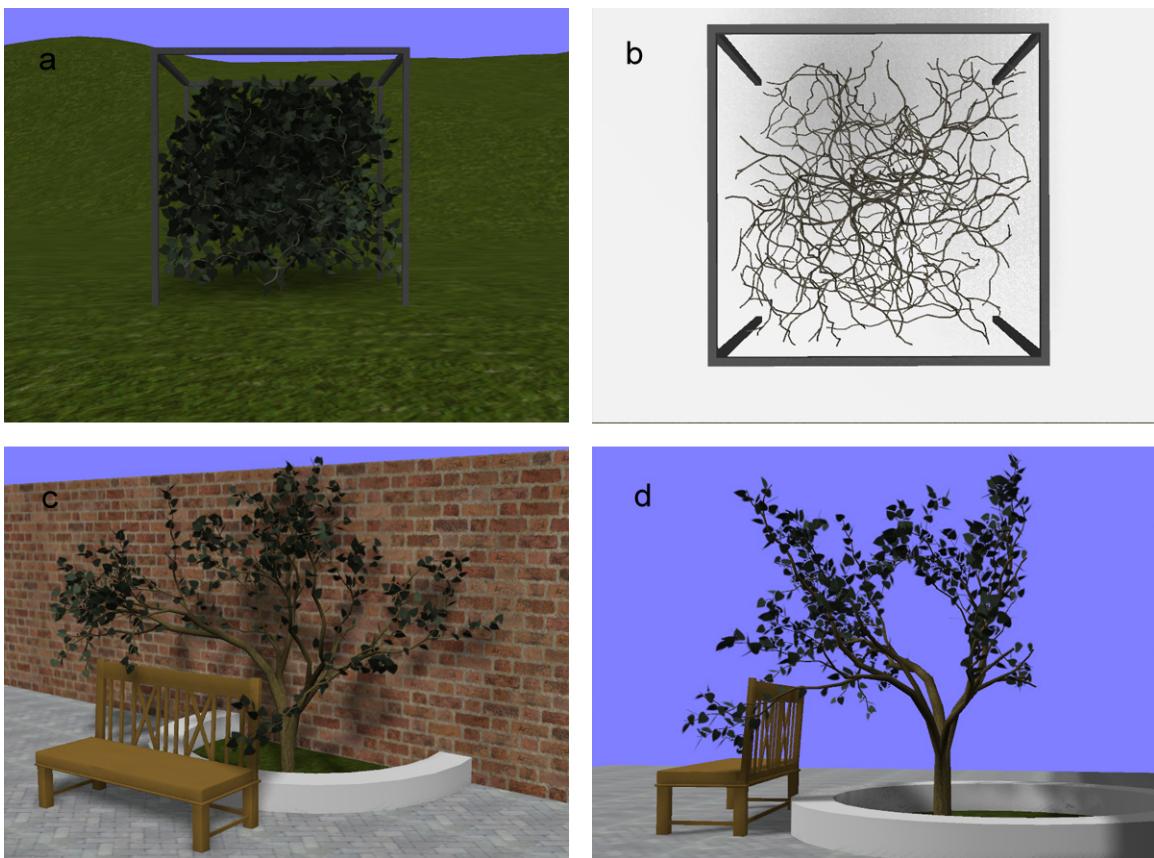
Levý i pravý kontext může stejně jako předchůdce mít své parametry, které je nutné načíst pro zpracování výrazů. Pozice modulů, získané jako návratové hodnoty během ověřování kontextů, slouží právě pro získání těchto parametrů. Díky rozšíření o zpracování kontextu mohl být do zásuvného modulu implementován také mechanismus pro zpracování dotazů. Podrobnějším popisem implementace dotazů se zabývá následující kapitola 5.6.

Aplikaci dotazů i signálů pomocí kontextových L-systémů lze pozorovat na obrázcích 5.4. Účelem bylo vytvořit modely, jež pomocí určité funkce mohou reagovat na své okolí. Reakce v tomto případě spočívá v ukončení růstu a šíření signálu o přerušení zpět směrem ke kořenům. Pokud se při šíření signálu nalezne místo, kde je možné vytvořit větvění, větev se rozdělí. Nově vytvořená větev pak pokračuje v růstu. Lze takto simulovat jev, který se obdobně vyskytuje i v přírodě. Pokud je totiž z nějakého důvodu zamezeno věti dále růst, aktivuje se pupen na cestě ke kořenům. Zmíněné obrázky ukazují dva způsoby použití tohoto jevu. V případě obrázků (a) a (b) se jedná o simulaci zastříhování živého plotu. Jakmile větev opustí vymezený prostor, je zastřížena a růst pokračuje jinde. Obdobně funguje i druhý model na obrázcích (c) a (d). Zde je tímto mechanismem znemožněno prorůstání stromu do zdi.

5.6 Dotazy

Díky dotazům lze již během generování zjistit, kde a v jaké poloze se bude daný modul nacházet v prostoru. Rozšiřitelnost knihovny FunctionParser umožňuje do výrazů vkládat funkce, které pak mohou vrátit kterýkoli parametr polohy želvy. Toto připojení uživatelsky definovaných funkcí je prováděno pouze do výrazů, které tyto funkce obsahují. Je to z důvodu, že výrazy, které tyto funkce obsahují, nemohou být optimalizovány vnitřními algoritmy knihovny FunctionParser. Tyto dotazovací funkce jsou ve skutečnosti statickými metodami třídy Query. Každá tato metoda je bezparametrická a vrací hodnotu double. Pro každou souřadnici každého vektoru polohové matice želvy existuje jedna metoda.

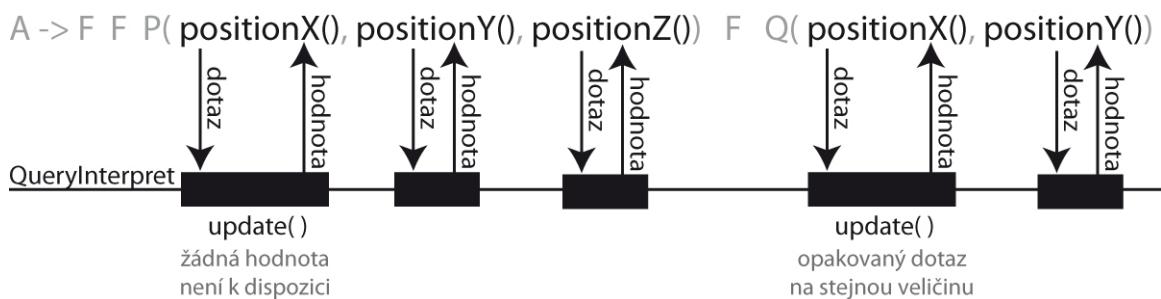
Třída Query slouží jako adaptér mezi třídami FunctionParser a QueryInterpret.



Obrázek 5.4: Stochastic parametric 2L-systems.

Třída `QueryInterpret` je speciálně uzpůsobený interpret pro zpracování dotazů. Jeho fungování je podobné jako u třídy pro interpretaci geometrie, jež bude popsána v kapitole 6.2. Na rozdíl od ní však nevytváří žádnou geometrii a interpretuje jen symboly, které jsou důležité pro polohu želvy. Je implementován dle návrhového vzoru *jedináček*.

Na základě volání některé z metod třídy `Query` pak dostane pokyn k interpretaci slova. Interpretována je vždy jen ta část slova, která předcházela právě zpracovávanému modulu s dotazy. Nakonci interpretace je poloha želvy uložena a připravena pro dotazovací metody. Celá tato operace získání polohy je zajištěna vnitřní metodou `update()`. Problémem je, že hodnoty polohy želvy velice rychle stárnou a jejich obnovení je kvůli nutnosti interpretace řetězce časově náročná procedura. Pro omezení volání metody `update()` je implementován jednoduchý algoritmus. Pokud je některá souřadnice přečtena podruhé, předpokládá se, že se jedná o jiný modul a je tedy třeba provést obnovu. Na obrázku 5.5 lze vidět, jakým způsobem vypadá pravidlo s dotazy a jak se jednotlivé dotazy pomocí instance třídy `QueryInterpret` vyhodnocují. U dotazů, které si využijí provedení interpretace slova je spolu s důvodem uvedena metoda `update()`. Ostatní dotazy obdrží odpověď okamžitě.



Obrázek 5.5: Zpracování dotazů třídou `QueryInterpret`

Instance třídy `QueryInterpret` tedy při prvním volání dotazu `positionX()` zjistí všechny souřadnice polohy želvy. Poté je možné vracet jednotlivé souřadnice polohy. Jakomile se jeden z dotazů zopakuje, provede se obnova polohy želvy a tedy i nová interpretace slova. Stejným způsobem reaguje interpret na dotazy týkající se aktuálních směrových vektorů želvy.

Zpracováním dotazů lze tedy do řetězce slova dostat poziční hodnoty ještě před vykreslovací fází. Kombinací s kontextovými pravidly s podmínkou lze dosáhnout toho, že některá pravidla budou použita jen při určitých hodnotách těchto dotazů. Druhou možností je využít tyto hodnoty ke změně parametrů modulů. Například v případě popínavých rostlin lze na základě polohy a směru rostoucí větve vyhodnotit, kterým směrem by se měla daná větev vyvíjet a podle toho pak ovlivnit její směr.

5.7 Podřízené L-systémy

Problematiku návrhu modelu pomocí L-systémů lze rozdělit na několik úrovní. Pro lepší kontrolu, přehlednost a také nastavitelnost je lepší jednotlivé dílčí celky hierarchicky seřadit. U botanických organismů se může jednat například o rozdělení na rostlinné orgány. Jde však samozřejmě jít i hlouběji. Výhodou tohoto přístupu je modularita. Každý takovýto dílčí L-systém má svá vlastní nastavení a může být použit v libovolném počtu jiných L-systémů. Pokud je tedy například vytvořen L-systém pro list, je velice jednoduché přidat jej jako podsystém ke všem L-systémům stromů stejného druhu.

Každý L-systém si své podsystémy spravuje sám. Po jejich načtení pomocí metod třídy `AbstractFile` je každý L-systém inicializován a zpracován podobně jako ve třídě `LSystemGenerator`. Každému podsystému je přiřazeno unikátní číslo. Pod ním je tento systém reprezentován v řetězci pomocí modulu `# (L-System_ID)`. Všechna slova jsou vygenerována a uložena v nadřazeném L-systému. Do řetězce jsou vložena až po ukončení iteračního procesu, kdy jsou všechny moduly `# ()` nahrazeny vygenerovanými slovy. Na začátku a konci každého slova podsystému jsou speciální moduly `$`, jež poté informují interpret o změně L-systému. Ten pak dle parametrů tohoto modulu přepne na zpracování řetězce dle odpovídajících nových parametrů. Neboť je toto chování dosti podobné zpracování závorek, věnuje se jím

5.7. PODŘÍZENÉ L-SYSTÉMY

společná kapitola o implementaci zásobníku **6.2.1**.

Kapitola 6

Interpretace slov

Předchozí kapitola se věnovala výhradně iteračnímu procesu L-systémů za účelem vytvoření slova, jež bude reprezentovat požadovaný model. V této kapitole budou podrobněji rozvedeny techniky, které takto vygenerované slovo interpretují a vytváří jeho geometrickou podobu. Pro interpretaci byla zvolena výše zmíněná technika želví grafiky. Jednotlivé moduly pak slouží jako příkazy želvě k různým úkonům. Díky abstrakci na úrovni modulů lze vybírat z různých technik, kterými želva kreslí, a tak vytvářet rozmanitější modely.

Kromě samotné geometrie je nutné řešit i jiné otázky týkající se zobrazení modelu. Jedná se zejména o mapování textur a o úroveň detailů. Na této vrstvě je také možné přidat i různé vlivy okolí. Jelikož se jedná pouze o interpretaci, tyto vlivy nijak nemohou ovlivnit strukturu výsledného modelu. Mohou však globálně ovlivnit celý model. Lze tak jednoduše přidat modelu závislost na některých silách jako je gravitace. Jinou možností je přiblížení rostlin svým reálným předlohám pomocí malé náhodnosti úhlů. Modely tak ztratí svou viditelnou matematickou přesnost.

Jako rozhraní pro interpretaci slov slouží abstraktní třída `AbstractInterpret`. Implementováni jsou dva potomci. `TurtleInterpret` zpracovává všecké příkazy želvy a slouží k vytváření geometrie. `QueryInterpret` provádí zpracování dotazů ohledně polohy želvy. Tento interpret zpracovává jen vybrané příkazy. Podrobněji byl popsán v kapitole 5.6.

Řetězec generovaný L-systémem je přetyповán na nový typ. Z instance třídy `LongString` je převeden na instanci třídy `ParseableString`. Jedná se o jednoduchou třídu, která obsahuje metody pro dopředné procházení řetězce a získávání modulů a jejich parametrů.

6.1 Spojení s grafem scény

Pro zobrazení generované geometrie je důležité napojení na graf scény. Při vytváření interpretu je mu konstruktorem předán ukazatel na rodiče v grafu. Vzhledem k tomu, že slovo může díky mechanismu podsystémů obsahovat řetězce několika L-systémů mající různé parametry, je nutné i při vytváření geometrie tyto L-systémy oddělit. Díky grafu scény je k dispozici ideální řešení. Pro každý L-systém vytvoří interpret jeden uzel s geometrií, jež bude obsahovat i svá nastavení. Tyto uzly jsou instancemi třídy `LSGeode`. Tato třída je rozšířením třídy `osg::Geode`. Navíc obsahuje metody pro volbu správného typu želvy a také metody pro předání výchozích hodnot nastavení želvy. Všechny tyto hodnoty jsou získány z parametrů, jež se načítají při zpracování souborů L-systémů potomky třídy `AbstractFile`.

6.2. INTERPRETACE POMOCÍ ŽELVÍ GRAFIKY

Při inicializaci instance `LSGeode` dochází také k nastavení uzlu. Nastaví se textura, materiály a další vlastnosti. Veškeré parametry jsou zpracovány a konvertovány do takové podoby, aby při generování byly bud' již nastaveny nebo rychle připraveny ke zpracování.

6.2 Interpretace pomocí želví grafiky

Úkolem interpretu je projít celé slovo typu `ParseableString` a moduly, které rozpozná, převést na příkazy. Pro tento převod slouží metoda `parse()`. Každá třída interpretu obsahuje v této metodě svou sadu příkazů, které umí interpretovat. Následně pak volá metody pro manipulaci s želvami na zásobníku nebo nastavuje a kreslí želvou, která je na vrcholu zásobníku.

6.2.1 Zásobník pro ukládání želvích instancí

Pro zpracování slov, které obsahují závorky a podřízené L-systémy, je nutná dobrá implementace zásobníku pro želvy. K tomuto účelu slouží třída `TurtleStack`, která poskytuje standardní funkce zásobníku. Je však upravena pro použití v tomto zásuvném modulu. Při interpretaci se pracuje vždy pouze se želvou, která je aktuálně na vrcholu zásobníku. Existují proto speciální moduly, jež dokáží s obsahem zásobníku manipulovat.

- [Tato závorka označuje počátek nové vedlejší větve. V tomto stavu je nutné vytvořit metodou `push()` na zásobníku novou želvu, která se vykreslováním této větve bude zabývat. Parametry i typ si tato želva nastaví podle dosavadní želvy na vrcholu zásobníku.
-] Ukončení vedlejší větve je signalizováno tímto symbolem. Želva je z vrcholu zásobníku odstraněna a další geometrii generuje želva pod ní. V růstu tedy pokračuje hlavní větev.
- \$**(x)** Jednotlivé L-systémy mají již od svého načtení přiřazen jednoznačný identifikátor. Symbol dolaru s jedním parametrem oznamuje interpretu, že má dojít ke změně L-systému. Podle identifikátoru v parametru modulu je vybrán správný L-systém a jeho odpovídající uzel s geometrií `LSGeode`. Ten už poté ví, který typ želvy použít a jaké výchozí parametry jí nastavit. Některé parametry, jako je matice polohy, mohou být nastaveny podle dosavadní želvy na vrcholu zásobníku. S nově vytvořenou želvou se pak na zásobníku provede operace `push()`.
- \$ Ukončení slova podsystému signalizuje symbol dolaru bez parametru. Z vrcholu zásobníku je odstraněna želva a ve vykreslování pokračuje želva nadřazeného L-systému.

6.2.2 Želví rozhraní

Pro větší modularitu byl při návrhu zvolen model, jenž umožňuje použití libovolného množství typů želv. Lze tedy generovat různou geometrii pomocí stejných příkazů. Všechny třídy želv dědí z abstraktní třídy `AbstractTurtle`. Tato třída obsahuje rozhraní pro úplnou kontrolu želvy. Většinu tvoří metody pro zpracování příkazů, které se ve slovech vyskytují jako moduly. Kromě toho obsahuje metody k propojení s uzlem geometrie. Do něj pak přímo ukládá veškerou geometrii. K propojení dochází při vkládání želvy na vrchol zásobníku. Je zde k dispozici i několik metod pro vykreslování pomocné geometrie při ladění.

Každá želva má celou řadu parametrů, jež se mohou během vykreslování neustále měnit. Proto obsahuje strukturu `TurtleProperties`, která všechny tyto hodnoty uchovává. Parametry si želvy mezi sebou předávají při operacích na zásobníku. Pokud se jedná o první želvu L-systému, nahrají se výchozí parametry z instance třídy `LSGeode`.

6.2.3 Želví příkazy

Každá želva interpretuje celou řadu příkazů. Ne všechny želvy musí nutně implementovat všechny příkazy. Některé mohou být úzce specializované. Obecně lze příkazy rozdělit do několika kategorií.

- *Základní pohybové příkazy* — Většina želv má implementovány základní příkazy pro pohyb. Metody pro pohyb jsou společné a nachází se v abstraktní třídě `MovingTurtle`. Umožňují například pohyb želvy dopředu a natáčení kolem všech os. Želva se může pohybovat i bez generování geometrie pouze za účelem přemístění se.
- *Pokročilé pohybové příkazy* — Jedná se o nestandardní pohyby želvy, jako je otočení do protisměru nebo vyrovnaní náklonu do vodorovné polohy.
- *Změna stavových proměnných* — Řadu parametrů želvy lze během interpretace měnit. Je tak možné například prodloužit délku kroku nebo měnit výchozí úhel rotace.
- *Nastavení vnějších vlivů* — Některé příkazy mohou ovlivnit reakci želvy na globální vnější vlivy. V případě gravitace se může například jednat o změnu pružnosti větve.

6.3 Generování geometrie

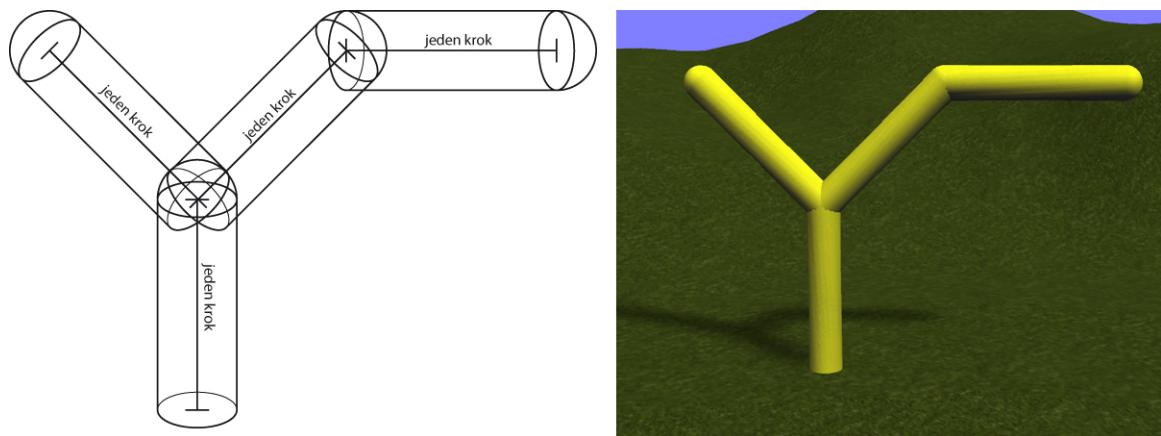
Z definice želvy vyplývá, že ke kreslení dochází pouze při jediném typu příkazu. Jedná se o krok vpřed. V případě trojrozměrné implementace v zásuvném modulu `LSystem` dochází při interpretaci tohoto příkazu ke generování geometrie. Implementačně se jedná o metodu `drawStep()` deklarovanou ve třídě `AbstractTurtle`. Její konkrétní definice záleží na typu želvy. Může se jednat o planární i trojrozměrné objekty, nahrané modely a jiné. Veškerá geometrie se ukládá do jedné či více instancí třídy `osg::Geometry`, která slouží jako úložiště veškeré geometrie instance třídy `LSGeode`. Každá želva má uložen ukazatel na tyto

6.3. GENEROVÁNÍ GEOMETRIE

kontejnery s geometrií. Díky tomuto propojení uzelů geometrie s želvami na zásobníku je pak geometrie vykreslena dle parametrů jednotlivých L-systémů.

6.3.1 Válce s klouby

Jedná se o jednoduché vykreslování grafiky, jež simuluje želvou kreslenou čáru. Pro každý krok se vygeneruje právě jeden válec o předdefinovaném poloměru a délce jednoho kroku. Umístěn je tak, aby přímka procházející středem obou jeho podstav byla totožná s přímkou která prochází krajinmi body kroku želvy. Protože by při rotaci mezi jednotlivými kroky docházelo k nespojitostem v geometrii, umisťují se na podstavy válců koule, které plní funkci kloubu. Při shodném poloměru s válci nespojitosti vyplní. Pro optimalizaci je při implementaci použita pouze polokoule. Lze také nastavit detail jednotlivých těles pomocí parametru `ContourDetail`.

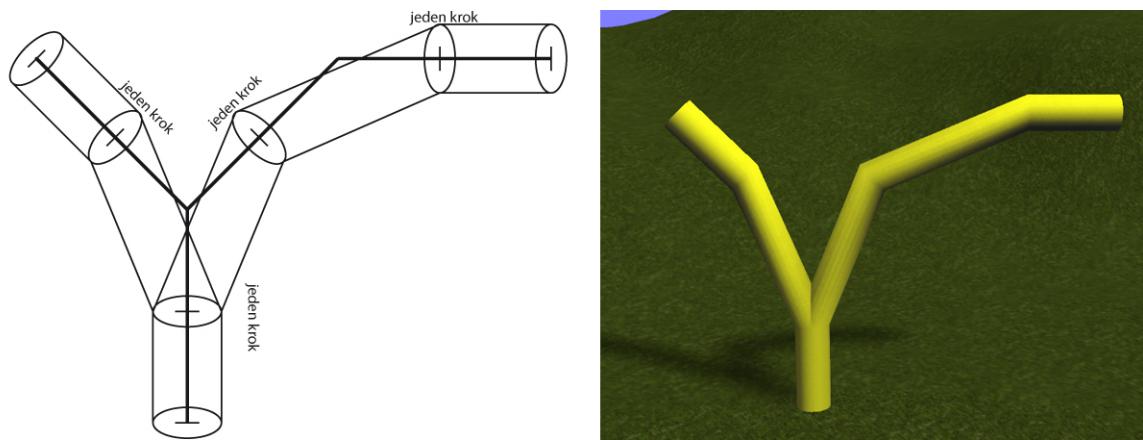


Obrázek 6.1: Generování válců s klouby.

6.3.2 Spojité válce

Želvy tohoto typu generují, podobně jako předchozí typ geometrii, složenou z válců. Díky odlišnému přístupu však zde nedochází k nespojitostem geometrie při rotacích želvy mezi jednotlivými kroky. Každý krok je zde rozdělen na dva půlkroky. Po provedení prvního půlkroku je pozice želvy uložena jako kontrolní bod a je použita jako bod pro generování kontury, která tvoří horní podstavu předešlé válcovité geometrie a zároveň spodní podstavu geometrie kroku, který bude případně následovat. Rovina kontury je vždy kolmá na směr pohybu želvy.

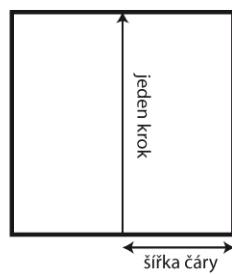
Geometrie v tomto případě tedy přesně nekopíruje trasu želvy. Generuje však vizuálně mnohem lepší výsledek než v případě válců s klouby a navíc kvůli absenci polokoulí obsahuje menší množství trojúhelníků. V tomto případě lze také mnohem lépe vyřešit navazování textur.



Obrázek 6.2: Generování spojité válcovité geometrie.

6.3.3 Obdélníky

Předchozí dva typy želv jsou vhodné pro tvorbu stonků a kmenů. Pro modely stromů je implementována tato třída pro tvorbu listů. Stromy obsahují velké množství tvarově málo rozmanitých listů. Právě pro tento účel je vhodná želva typu Rectangle. Při kroku vpřed vytváří vždy pouze obdélník. Tento obdélník je pak vhodný pro nanášení textury, jež díky průhlednosti listu jednoduše a tvarově přesně vykreslí. Velikost obdélníku se řídí běžnými parametry želvy. Jde o délku kroku a tloušťku čáry.



Obrázek 6.3: Generování geometrie pro listy.

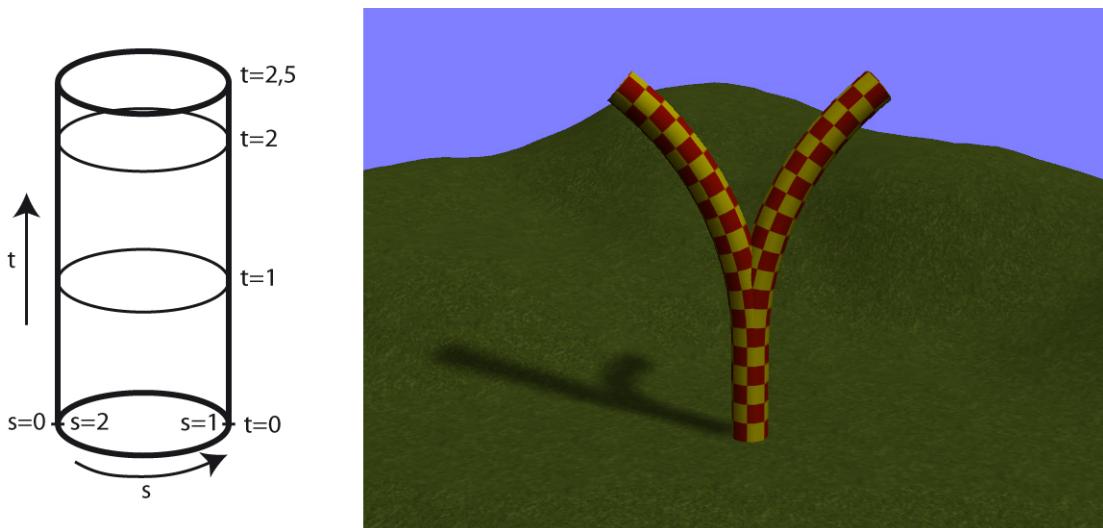
Listy stromů je vhodné vytvořit pomocí samostatného podřízeného L-systému. Pokud je pro interpretaci slov listů použita tato želva, nemusí být L-systém, jenž toto slovo generuje, nijak složitý. Většinou postačí jeden krok kupředu společně s případnými rotacemi.

6.4 Textury

Pro reálné zobrazení rostlinných organismů je důležité nanášení textur. U generované geometrie je tedy potřebné dbát na správné generování souřadnic textury. Zásuvný modul LSystem umožňuje nanášení difuzních textur na navazující válcovité a jednoduché planární objekty. Je tedy možné použít texturu jak pro geometrii větví, tak pro geometrii listů.

6.4.1 Mapování textur na navazující válce

Na geometrie větví se obvykle nanáší textury kůry nebo stonku, které jsou vytvořeny tak, aby při jejich skládání nebyly viditelné okraje textur. Z důvodu válcovitého tvaru se generování texturových souřadnic vždy řídí podle obvodu podstav válce. Aby textury správně navazovaly, je nutné, aby se po obvodu opakovaly v celočíselných násobcích. Tento násobek je možné nastavit parametrem `TextureSRepeatings`, který nastavuje, kolikrát se bude textura podél obvodu opakovat.



Obrázek 6.4: Mapování textury na válce

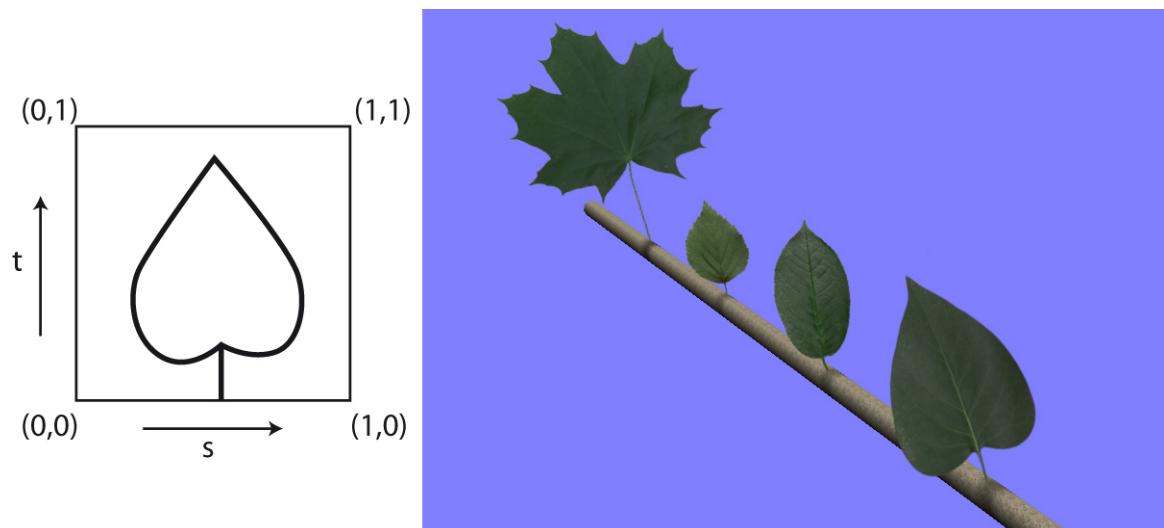
Směr podél obvodu odpovídá texturové souřadnici s . Směr pohybu želvy kupředu pak určuje směr texturové souřadnice t . Tato souřadnice se vždy přizpůsobuje obvodu větve tak, aby byl přibližně zachován poměr stran a textura tak působila přirozeně. Opakování textury v tomto podélném směru tedy nemusí být celočíselné a hodnota souřadnice z konce jednoho válcovitého dílu se přenáší dále jako počáteční hodnota pro další krok. Přenáší se však pouze desetinná část, neboť pouze ta je důležitá pro korektní navázání textury. Podobný princip nanášení textury lze nalézt i v práci [10]. Implementace v tomto zásuvném modulu navíc umožňuje opakování ve směru s . Způsob a příklad generování texturových souřadnic je znázorněn na obrázku 6.4. Celý válec představuje jeden krok želvy. Kolem obvodu se zde

6.5. MINIMALIZACE PŘÍČNÉHO NÁKLONU

textura opakuje dvakrát. Podélná textura je pak adekvátně přizpůsobena.

6.4.2 Mapování textur na obdélníky

Pro nanášení textury na objekty listů je použita jednoduchá metoda mapování. Jelikož je každý list reprezentován jedním obdélníkem, jeho krajní body odpovídají krajním bodům textury. Tvaru listu je dosaženo pomocí průhlednosti textury. Díky tomu je zapotřebí mnohem méně geometrie. Toto řešení však s sebou nese jedno úskalí. *OpenSceneGraph* obsahuje pro správné zobrazení průhledných objektů speciální koš. Při vykreslování postupně vybírá z koše jednotlivé objekty typu `osg::Geometry` seřazené podle vzdálenosti od pozorovatele od zadu dopředu. Díky tomu jsou i průhledné objekty vykresleny korektně. Kvůli optimalizaci je však vhodné mít všechny objekty listů v jedné geometrii, což však může mít za následek špatné seřazení objektů. Tato optimalizace tak může někdy způsobovat na okrajích listů *halo* efekt. Pro odstranění tohoto efektu je možné nastavit parametr `SeparateGeometryForTranslucency`, který vynutí použití oddělených geometrií pro každý objekt listu. Listy jsou v tomto módu korektně seřazeny. Vzhledem k velkému počtu instancí třídy `osg::Geometry` však dochází ke snížení výkonu.



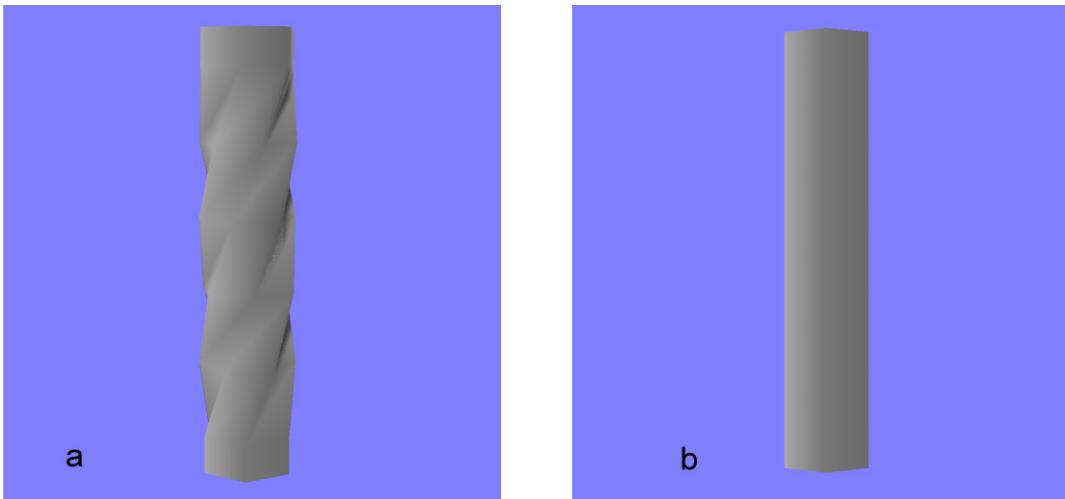
Obrázek 6.5: Ukázka mapování textur listů.

6.5 Minimalizace příčného náklonu

Interpretace želví grafikou je bezesporu jednou z velmi univerzálních metod pro interpretaci L-systémů. Přináší však s sebou i možné nevýhody. Jednou z nich je deformace válců při provádění rotace kolem osy \vec{H} . V každém kroku želva generuje body válcovité geometrie

6.6. ODEZVA NA SMĚROVÉ PODNĚTY

podle své kontury a řídí se svou aktuální orientací. Poloha bodů závisí na orientaci vektoru \vec{L} . Pokud v jednom kroku dojde k příčnému náklonu o příliš velký úhel, může dojít k deformaci geometrie.



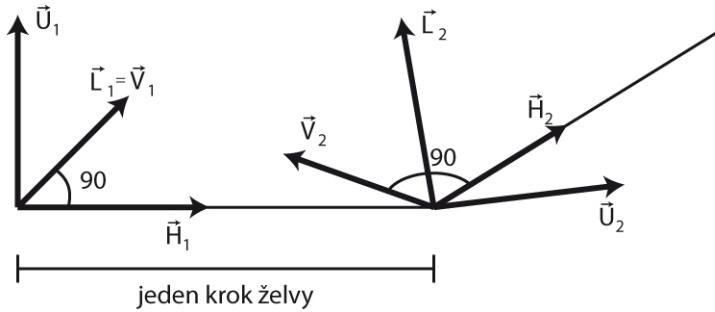
Obrázek 6.6: Porovnání vzhledu geometrie bez (a) a s (b) použitím minimalizace příčného náklonu při interpretaci řetězce $F/F/F/F/F/F$.

Někdy může být tento efekt žádoucí, většinou je ale lepší jej odstranit. Řešením může být jeden z několika algoritmů pro minimalizaci příčného náklonu. Pro zásuvný modul *LSystem* byla zvolena Bloomenthalova metoda minimalizace rotace, kterou Bloomenthal obdobně použil při generování svého modelu *Mohutný javor*. Kromě polohové matice želvy se zavádí navíc matice kontury. Tato matice pak slouží ke generování bodů kontury a vytvoření válcovité geometrie, která nepodléhá příčnému náklonu. Je vytvořena na základě ortogonální báze vektorového prostoru $(\vec{H}, \vec{V}, \vec{H} \times \vec{V})$, kde \vec{H} je směrový vektor želvy a \vec{V} je vektor, podle něhož se bude generovat geometrie. Tento vektor je před interpretací slova nastaven jako shodný s vektorem \vec{L} .

Při každém kroku želvy kupředu jsou v metodě `adjustMatrices()` upraveny hodnoty vektorů. Pokud označíme \vec{H}_1 jako směrový vektor želvy původní báze a \vec{H}_2 označíme směrový vektor želvy po vykonaném kroku vpřed, můžeme novou bázi získat otočením původní ortogonální báze kolem osy $\vec{H}_1 \times \vec{H}_2$ o úhel, který svírají vektory \vec{H}_1 a \vec{H}_2 . Vektor \vec{V} pak bude vždy ve stejné rovině s vektorem \vec{L} . Na rozdíl od něj však \vec{V} nebude podléhat příčnému náklonu.

6.6 Odezva na směrové podněty

Rostlinné organismy v přírodě reagují na celou řadu podnětů. Některé podněty, jako je gravitace, mohou mít na vzhled rostliny zásadní vliv a implementací reakce na tyto podněty lze



Obrázek 6.7: Princip minimalize příčného náklonu u dvou následujících bází polohy želvy.

modely podstatně přiblížit realitě. Pro směrové podněty se používá termín *tropismus*. Jedná se o změnu orientace rostlinného organismu na základě nějakého podnětu či podráždění [2]. Tento podnět je v případě tropismu veden vždy v nějakém směru. V přírodě existuje mnoho druhů tropismu. Některé z nich jsou velmi důležité i zajímavé pro simulaci.

Geotropismus/Gravitropismus V rámci simulace biomechanických jevů se jedná o nejdůležitější formu tropismu. Geotropismus je reakce rostlinného organismu na gravitaci. Příkladem může být směrování větví pod svou tíhou k zemi nebo naopak růst stonku proti gravitaci.

Fototropismus Dalším často simulovaným jevem je reakce na světelné podmínky, kdy se orgány rostlin orientují směrem k největšímu zdroji světla. Obdobnou formou je pohyb za sluncem nazývaný heliotropismus.

Při aplikaci těchto vlivů na želvu, dochází při každém kroku želvy ke korekci jejího směru tak, aby respektovala síly, které na ní působí.

6.6.1 Geotropismus

Pro implementaci tropismu byla použita metoda založená na výpočtu zrychlení momentu síly. Lze tak vypočítat úhel, o který je třeba otočit směrový vektor želvy \vec{H} směrem k vektoru podnětu \vec{T} .

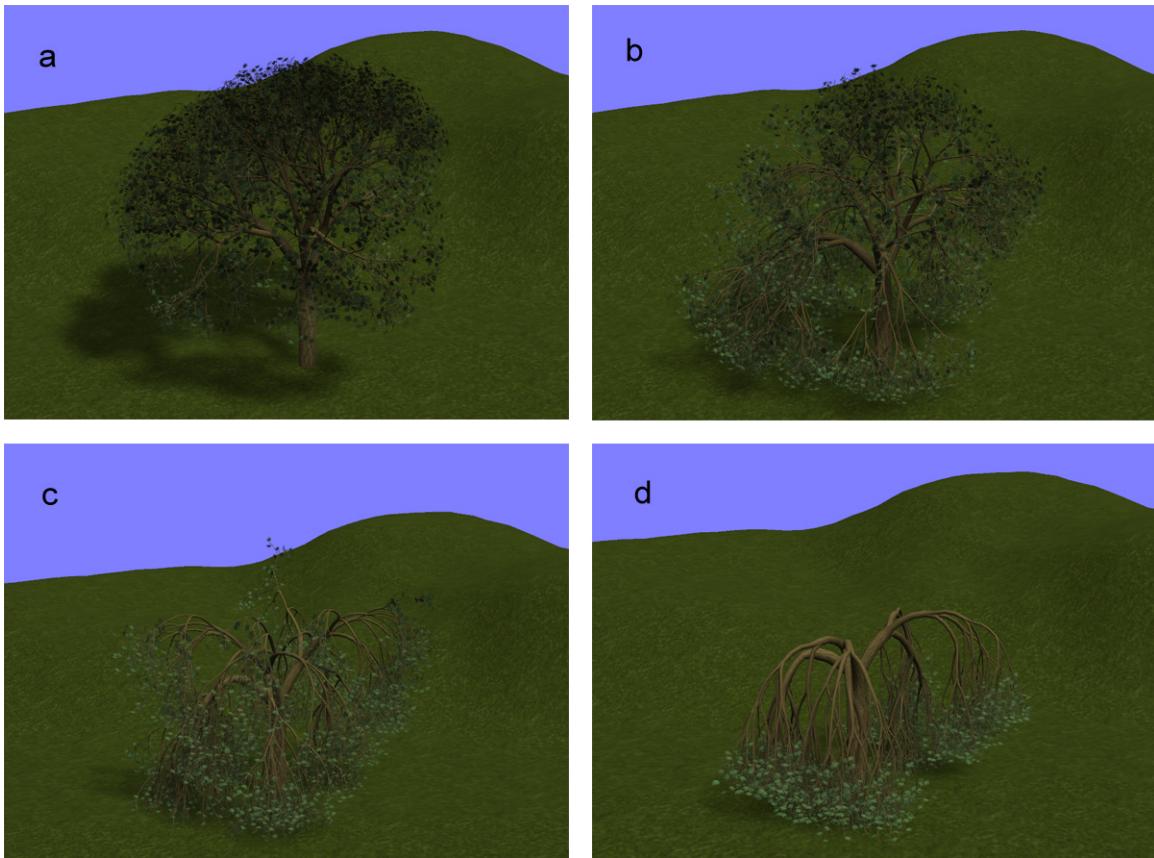
$$\beta = e |\vec{H} \times \vec{T}|$$

Rovnice 6.6.1: Výpočet korekčního úhlu při geotropismu.

Důležitou roli zde hraje parametr e , který udává přizpůsobivost danému podnětu. Například u větví tento parametr udává jejich pružnost. V zásuvném modulu je geotropismus

6.6. ODEZVA NA SMĚROVÉ PODNĚTY

implementován odděleně od ostatních tropismů, neboť se jedná o velmi důležitý podnět. Úhel podnětu směřuje vždy směrem k zemi (proti směrovému vektoru \vec{H} na počátku interpretace). Nastavení se provádí pomocí parametru `GeotropismElasticity`.



Obrázek 6.8: Ukázka vlivu geotropismu na ternární strom. Při generování stromů na obrázcích a, b, c a d byly použity hodnoty pružnosti 0.0, 0.1, 0.2 a 0.3.

Při každém kroku želvy je metodou `adjustMatrices()` vypočten pomocí výše zmínovaného vzorce nový směrový vektor a zbývající ortogonální vektory jsou adekvátně dopočítány.

6.6.2 Diatropismus

Tento vzorec lze dále rozšířit na obecnější formu 6.6.2. Jedná se o takzvaný diatropismus, při němž se rostlinné orgány snaží se směrovým vektorem podnětu svírat úhel γ . Tato obecná forma tropismu lze využít pro modelování různých jevů. Na obrázku lze vidět ovlivnění stromu při různých nastaveních. Implementace je obdobná jako u geotropismu. Navíc jsou však dispozici parametry `TropismAngle` pro nastavení úhlu γ a `TropismVector` pro na-

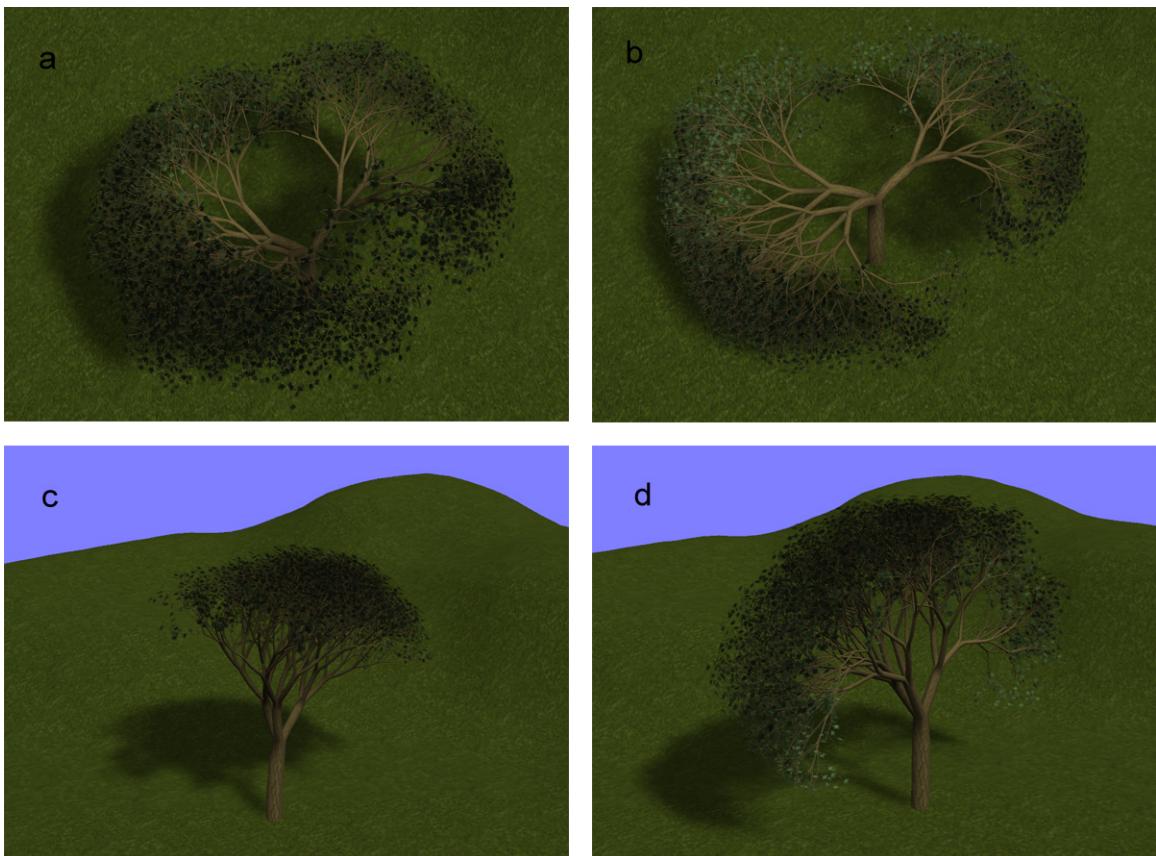
6.7. FLUKTUACE ÚHLŮ

stavení úhlu tropismu. Pružnost se zde nastavuje parametrem `TropismElasticity`.

Pro každý model lze nastavit maximálně jeden takový podnět. Zároveň však díky oddělené implementaci lze vždy zároveň nastavit i geotropismus.

$$\beta = e \left(\cos(\gamma) - \sin(\gamma) \frac{\vec{H} \cdot \vec{T}}{|\vec{H} \times \vec{T}|} \right)$$

Rovnice 6.6.2: Výpočet korekčního úhlu při diatropismu.



Obrázek 6.9: Stromy na obrázcích jsou generovány s aplikovaným plagiortopismem dle parametrů v tabulce 6.6.2.

6.7 Fluktuace úhlů

Jedním z velkých kladů L-systémů je, že každé slovo přesně odpovídá generované rostlině. Někdy je však žádoucí, aby bylo možné pomocí minimálních změn vytvořit organismus, který bude topologicky shodný, ale vizuálně mírně odlišný. Tímto způsobem lze z jednoho

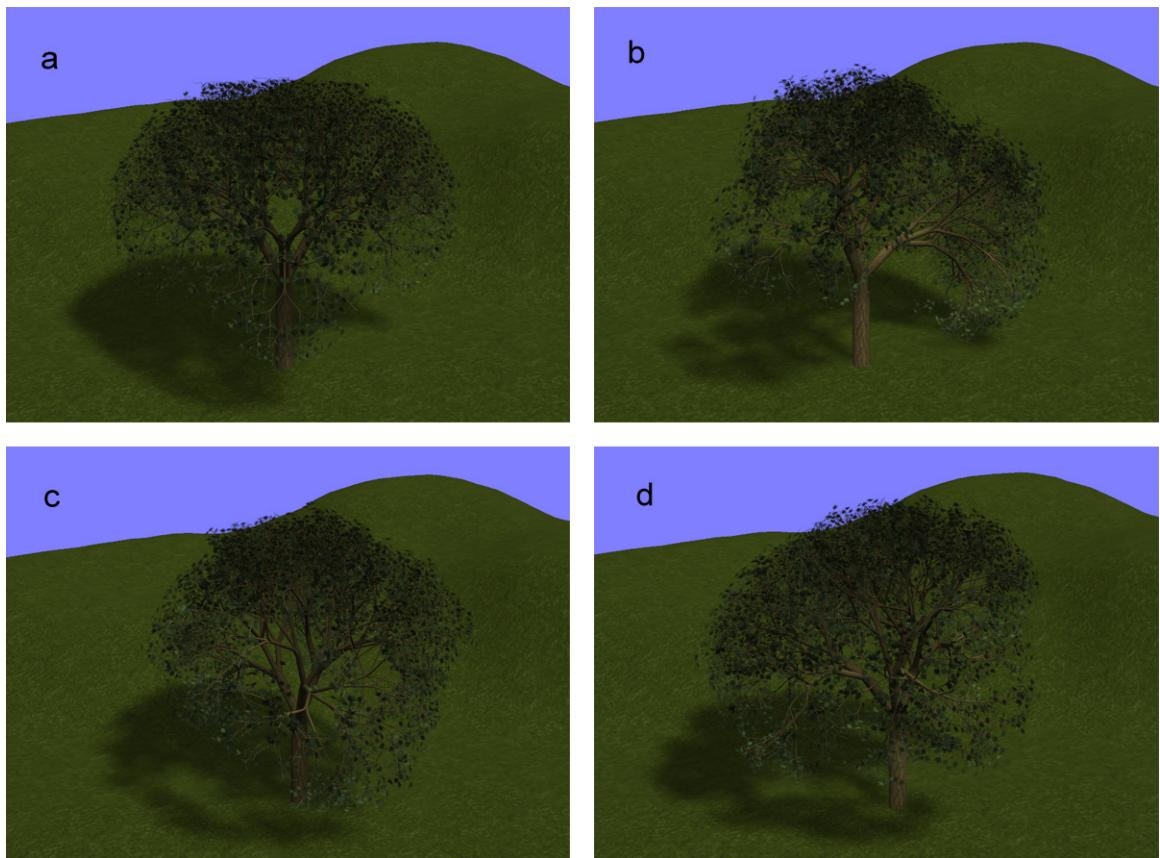
6.7. FLUKTUACE ÚHLŮ

Obrázek	Pružnost	Směrový vektor diatropismu	Úhel diatropismu
a	0.17	(0,0,1,0,0,0)	60
b	0.2	(0,0,1,0,0,0)	90
c	0.09	(0.7,1,0,0,0,0)	0
d	0.2	(0,0,0,0,1,0)	90

Tabulka 6.1: Parametry diatropismu.

již vygenerovaného slova vytvářet ne zcela identické jedince. Druhým argumentem pro zavedení toho mechanismu je často absolutní matematická přesnost úhlů generovaných rostlin. Mírným náhodným roztažením úhlů lze odstranit viditelnou pravidelnost a dosáhnout tak mnohem lepších vizuálních výsledků.

Nastavením parametru `AngleVariance` lze toho efektu dosáhnou i v zásuvném modulu *LSystem*. Jde o hodnotu v procentech, která udává o kolik procent se mohou úhly maximálně změnit. Při provádění libovolné rotace želvy kolem jedné z jejich os, je pak úhel otočení vynásoben náhodnou hodnotou získanou na základě tohoto parametru. I při této náhodnosti lze nastavit náhodné semínko. Nastavením parametru `AngleFluctuationSeed` na stejnou hodnotu lze vytvořit vícekrát stejného jedince. Pokud tento parametr není nastaven v konfiguračním souboru ani v souboru L-systému, je jako semínko použit systémový čas.



Obrázek 6.10: Obrázek a je vykreslen bez fluktuace úhlů. Obrázky b, c a d mají fluktuaci nastavenou na 50% a semínko 1, 2 a 3.

Kapitola 7

Výkonné testy modulu LSystem

Při implementaci zásuvného modulu byl brán ohled na optimalizaci celého procesu. Modul je navržen tak, aby bylo možné vkládání stromů i jiných rostlin „za běhu“. Důraz byl kladen zejména na rychlosť generování slov. V přechozích kapitolách této práce se vyskytují obrázky několika vzorových modelů. Tabulka 7.1 obsahuje data naměřená při testování modulu LSystem. Pro testování byla použita následující počítačová sestava:

CPU Intel Core 2 Duo T9400 (2,53 GHz)

Operační paměť DDR2 PC5300 — 3072 MB

Čipová sada Mobile Intel GM965 Express

Grafická karta nVidia GeForce 9600M GT 256 MB

Obrázek modelu	t_1	L	I	M	R	Q	t_2	T
5.3 (a) — D0L-systém	0,024	2	14	157002	4372	0	0,630	28419
5.3 (a) — 0L-systém	0,061	2	14	157002	4372	0	0,628	28419
5.3 (a) — 2L-systém	0,074	2	14	157002	4372	0	0,633	28419
5.3 (d) — bez listů	0,053	1	25	110112	2693	0	0,092	340
5.3 (d) — s listy	0,055	2	25	110112	23491	0	0,187	5992
5.4 (c)	0,663	2	20	38753	14157	2883	0,107	47486
5.4 (a)	3,083	2	90	785429	226138	6552	0,168	172610

Tabulka 7.1: Výsledky testů výkonu zásuvného modulu LSystem.

Tabulka 7.1 obsahuje kromě času zpracování i množství zpracovaných dat a počet provedených, po výkonné stránce důležitých, operací. Veškeré časy byly vypočteny jako průměr z pěti měření. Jednotlivé sloupce mají následující význam:

t₁ Udává čas potřebný pro vygenerování slova v sekundách. Jedná se tedy o nahrání a zpracování L-systému a vykonání všech iterací.

L Počet použitých L-systémů.

I Počet iterací, jež byly provedeny na hlavním L-systému.

M Celkový počet zpracovaných modulů.

R Jedná se počet aplikací přepisovacího pravidla rozdílného od identity.

Q Počet zpracovaných dotazů.

t₂ Čas interpretace všech modulů a vytvoření geometrie modelu.

T Celkový počet vytvořených želv pro dotazy i geometrii.

Výsledky z prvních tří řádků tabulky 7.1 patří k totožnému L-systému. Jedná se o jednoduchý ternární strom generovatelný i D0L-systémem. Liší se pouze použitým algoritmem pro generování. První řádek obsahuje časy při generování modelu pomocí algoritmu pro D0L-systémy. Díky optimalizaci pro tento typ má tento algoritmus nejlepší výsledek při generování slova. Tento algoritmus je také automaticky zvolen při správném nastavení parametrů typu L-systému. Druhý řádek obsahuje hodnoty při generování totožného modelu stochastickým parametrickým 0L-systémem. Hodnoty v třetím řádku pak byly získány pomocí generování algoritmem pro stochastické parametrické 2L-systémy. I když v případě tohoto modelu není rozdíl pro člověka znatelný, při generování delších a komplikovanějších slov, či většího počtu L-systému, bude volba správného, pro daný účel optimalizovaného, algoritmu důležitá.

Dalším modelem byl strom vytvořený pomocí stochastického parametrického L-systému. Čtvrtý a pátý řádek tabulky nabízí srovnání generování modelu bez listů a s listy. Tento model obsahuje ve svých pravidlech výrazy, jež jsou při každém přepisu předchůdce na následníka vyhodnocovány. Zároveň také náhodně vybírá pravidla dle podmínek, jež jsou s každým modulem předchůdce různé. Poslední dva řádky obsahují hodnoty modelů kontextových L-systémů. Prvním z nich je simulace živého plotu. Druhý pak simuluje chování stromu u překážky. Oba tyto modely využívají kromě metod zmíněných u předchozích modelů také mechanismus dotazů pro zjišťování polohy želvy.

Kapitola 8

Závěr

Simulace přírody je, a v nejbližší budoucnosti nejspíš i bude, jednou z nejvíce zkoumaných oblastí počítačové grafiky. Poznat a napodobit jevy kolem nás se snaží odborníci již od počátku výpočetní techniky. Generování rostlinných organismů je pro řadu vědců velkou výzvou. Jedná se sice o geometricky složité organismy, jejich stavbu, vývoj a vlastnosti lze však často jednoduše matematicky popsat. Díky tomu se tímto tématem zabývá několik výzkumných skupin a k dispozici je vcelku velké množství literatury z oblasti počítačové grafiky i biologie.

V rámci této práce vznikl pro systém Vrecko jednoduše rozšířitelný zásuvný modul pracující s L-systémy, jež jsou v současné době nejvíce používanou metodou pro generování rostlinných organismů. Pro vývoj tohoto modulu bylo zapotřebí nastudovat množství literatury, která se zabývá různými metodami generování slov pomocí L-systémů a vytvářením geometrie rostlinných organismů. Nutností však bylo zaměřit se i na materiály zabývající se biologickou stavbou rostlin. Při návrhu byl pak kladen velký důraz na modulárnost a zároveň optimalizaci celého jádra zásuvného modulu. Byly proto navrženy specializované a rychlé datové struktury. Většina tříd je navržena proti rozhraní a tak není problém libovolnou funkcionality rozšířit. Pro srozumitelnost a rozšířitelnost kódu byla řada tříd navržena pomocí návrhových vzorů.

Zásuvný modul se skládá ze dvou samostatných celků. První z nich generuje slova pomocí uživatelsky zadaných L-systémů. Díky implementaci různých pokročilých typů L-systému, je možné vytvářet slova, jež simulují složité struktury a dodržují různé přírodní nebo i jiné zákonitosti. Výkonné testy ukázaly, že přepisovací proces je velice rychlý a nebrání tak ve vytváření slov v reálném čase. K většimu zpomalení nedochází ani při použití výrazů v parametrických L-systémech a náhodného výběru následníka dle zadaných kritérií. I kontextové L-systémy jsou navrženy tak, že je ověřování kontextu jednoznačné a dostatečně rychlé pro použití v reálném čase. Razantnější zpomalení přichází až s použitím dotazů. Zjišťování polohy želvy již v době generování slova je časově nejnákladnější procedurou, která však umožňuje reakci L-systému na okolí. Díky tomu byly vytvořeny modely jež se automaticky vytvářejí a zastřívají do parametricky definovaného tvaru. Uživatelský vstup se provádí pomocí strukturovaných XML souborů s L-systémy a jejich parametry.

Druhý samostatný celek se stará o generování geometrie pomocí želví grafiky. Je optimizován především pro tvorbu stromů, jež byly prvním cílem této práce. K dispozici jsou želvy, které generují geometrii kmenů, stonků i jednoduchých listů. Díky jednoduchým rozhraním je však jednoduše možné množství implementovaných typů želv rozšířit a vytvářet

libovolná jiná přírodní nebo umělá geometrická tělesa. I zde je navíc implementována řada rozšiřujících metod, které dělají modely zajímavějšími a reálnějšími.

Spojením těchto dvou částí vznikl modul, který umožňuje vytvářet velice širokou škálu modelů, které pak mohou být součástí jiných grafických scén či simulací systému Vrecko. Tato práce ukázala, jak zajímavé může generování rostlin být a snad také položila základní kámen generování rostlin v tomto interaktivním virtuálním prostředí.

Literatura

- [1] Abelson, H. a DiSessa, A.: *Turtle geometry*, M.I.T. Press, 1982. [3.1](#)
- [2] Bell, A. a Bryan, A.: *Plant form: an illustrated guide to flowering plant morphology*, Timber Press, 2008. [6.6](#)
- [3] Hanan, J.: *Modeling the mighty maple*, Computer Graphics (SIGGRAPH '94 Conference Proceedings), 1985.
- [4] Deussen, O. a Lintermann, B.: *Digital Design of Nature: Computer Generated Plants and Organics*, Springer-Verlag, 2005, 978-3-540-40591-7.
- [5] Hanan, J.: *Parametric L-systems and their application to the modelling and visualisation of plants*, , 1992. [2.4](#)
- [6] Holton, M.: *Strands, gravity and botanical tree imagery*, Computer Graphics Forum, 1994. [3.4](#)
- [7] Kelemenová, A.: *Complexity of L-systems*, Springer-Verlag, 1986. [3.2](#)
- [8] von Koch, H.: *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes*, Acta mathematica, 1905. [2.1](#)
- [9] Lindenmayer, A.: *Mathematical models for cellular interaction in development, Díl I a II*, Journal of Theoretical Biology, 1968. [2.2.3](#)
- [10] Měch, R. a Prusinkiewicz, P. a Hanan, J.: *Extension to the graphical interpretation of L-systems based on turtle geometry*, CSIRO Publishing, 1997. [6.4.1](#)
- [11] Mitchinson, G. a Wilcox, M.: *Rules governing cell division in Anabaena*, Nature, 1972. [2.2](#)
- [12] Prusinkiewicz, P. a Lindenmayer, A.: *The Algorithmic Beauty of Plants*, Springer-Verlag, 1990, 9780387972978. [2.5](#), [3.3](#), [3.5](#)
- [13] Prusinkiewicz, P. a Kari, L.: *Subapical bracketed L-systems*, Proceedings of the Fifth International Workshop on Graph Grammars and their Application to Computer Science, 1994. [2.3](#), [3.2](#), [3.4](#)
- [14] Prusinkiewicz, P. a Hammel, M. a Hanan, J. a Měch, R.: *L-systems: from the Theory to Visual Models of Plants*, CSIRO Publishing, 1996. [2.5](#)
- [15] Rozenberg, G. a Saloma, A.: *The mathematical theory of L-systems*, Academic Press, 1980. [2.2](#)
- [16] Žára, J. a Beneš, B. a Sochor, J. a Felkel, P.: *Moderní počítačová grafika*, Computer Press, a.s., 2004.

Příloha A

Přehled interpretovaných příkazů pro želví grafiku

V interpretační fází prochází instance některého z interpretů celý zpracovávaný řetězec modulů zleva doprava. Jednotlivé moduly pak převede na příkaz bud' některé z želv nebo přímo zásobníku s želvami. Tato příloha obsahuje přehled všech interpretovaných modulů, jejich identifikující znak a popis příkazu, který vykonají. Většina modulů umožňuje zadání parametru. Pokud není tento parametr zadán, použije se výchozí hodnota nastavená v konfiguraci modulu LSystem.

A.1 Rotace želvy

Želva může provádět rotace pouze kolem některého ze svých bázových vektorů.

+(α) Změní kurs rotací kolem osy U o úhel α doleva.

-(α) Změní kurs rotací kolem osy U o úhel α doprava.

&(α) Provede podélný sklon rotací kolem osy L o úhel α dolů.

^(α) Provede podélný sklon rotací kolem osy L o úhel α nahoru.

\(α) Provede příčný náklon rotací kolem osy H o úhel α doleva.

/(α) Provede příčný náklon rotací kolem osy H o úhel α doprava.

| Provede otočení do protisměru. Tento symbol odpovídá příkazu +(180) nebo -(180).

= Vyrovnaný příčný náklon do vodorovné polohy vůči původnímu souřadnicovému systému.

A.2 Změna vlastností želvy

Tyto příkazy mění hodnoty, jež jsou použity při bezparametrickém použití modulů.

- :**(x)** Vynásobí hodnotu výchozího úhlu hodnotou x .
- '**(x)** Vynásobí hodnotu výchozí délky kroku hodnotou x .
- !**(x)** Vynásobí hodnotu výchozí tloušťky kreslené čáry hodnotou x .

A.3 Změna pozice želvy

Tyto příkazy posouvají želvu kupředu ve směru vektoru \vec{H} .

- F(x)** Posune želvu o délku x kupředu. Tento příkaz generuje geometrii.
- Z** Posune želvu kupředu o poloviční délku než je výchozí hodnota. Tento příkaz generuje geometrii.
- f(x)** Posune želvu o délku x kupředu. Tento příkaz negeneruje geometrii.
- z** Posune želvu kupředu o poloviční délku než je výchozí hodnota. Tento příkaz negeneruje geometrii.

A.4 Operace na zásobníku želv

Větvení a vkládání podřízených L-systémů zajišťují příkazy pro manipulaci se zásobníkem želv.

- [Vloží na zásobník novou želvu, která sdědí parametry od želvy, jež byla na vrcholu zásobníku před ní.
-] Odstraní z vrcholu zásobníku želvu.

A.5. ZMĚNA NASTAVENÍ TROPISMU

#{ID} Vloží, a po interpretaci opět odstraní, želvu podřízeného L-systému. Parametr ID je unikátní identifikátor L-systému.

% Vynutí násilné ukončení větve. Odstraní všechny symboly dané větve, jež za tímto symbolem následují.

A.5 Změna nastavení tropismu

Pomocí těchto příkazů lze změnit pružnost želv.

; (x) Vynásobí hodnotu pružnosti při diatropismu hodnotou x.

~ (x) Vynásobí hodnotu pružnosti při geotropismu hodnotou x.

Příloha B

Přehled parametrů pro nastavení L-systémů

Zásvný modul LSystem se při generování a interpretaci slova řídí celou řadou parametrů. Jejich globální nastavení nezávislé na konkrétním L-systému lze provést v konfiguračním souboru. Parametry lze nastavit i pro konkrétní L-systém přímo v jeho souboru. Tato nastavení pak mají přednost před globálními.

B.1 Parametry generátoru slov

Iteration Při nastavení této hodnoty je po nahrání L-systému automaticky spuštěn iterační proces a provede se nastavený počet iterací. Hodnota je typu `unsigned integer`.

UseQueries Povolí použití dotazů. Hodnota je typu `unsigned integer`.

Ignore Výčet znaků, jež budou ignorovány při párování kontextu. Hodnota je řetězec znaků.

RandomSeed Nastaví semínko generátoru pseudonáhodných čísel. Pokud tento parametr není nastaven, použije se jako semínko systémový čas počítače. Tento parametr lze nastavit pouze v konfiguračním souboru jako globální hodnotu. V parametrech L-systému nebude mít žádný vliv. Hodnota je typu `unsigned integer`.

B.2 Parametry želvy

TurtleType Nastaví typ želvy, která bude interpretovat slovo.

Jedná se o výčtový typ a implementovány jsou tyto tři typy:

JOINTED_PIPE Vykresluje geometrii válců s klouby.

STRAIGHT_PIPE Vykresluje geometrii navazujících válcovitých těles.

RECTANGLE Vykresluje obdélníky. Vhodná pro jednoduché listy.

DefaultAngle Nastaví výchozí úhel pro veškeré rotace želvy. Hodnota je typu `double`.

DefaultLength Nastaví výchozí délku kroku želvy. Hodnota je typu `double`.

DefaultRadius Nastaví výchozí poloměr čáry, kterou želva kreslí. Hodnota je typu `double`.

AngleMultiplier Při bezparametrickém použití symbolu `:` je touto hodnotou vynásoben výchozí úhel. Hodnota je typu `double`.

LengthMultiplier Při bezparametrickém použití symbolu `'` je touto hodnotou vynásobena výchozí délka kroku. Hodnota je typu `double`.

RadiusMultiplier Při bezparametrickém použití symbolu `!` je touto hodnotou vynásoben výchozí poloměr kreslené čáry. Hodnota je typu `double`.

DegreesToRadians Při zadávání úhlů ve stupních musí být tato hodnota nastavena na 1.

B.3 Parametry vykreslované geometrie

ContourDetail Tento parametr udává úroveň detailu vykreslované geometrie. U extrudovaných těles se jedná o počet vrcholů kontury. Hodnota je typu `unsigned integer`.

AngleVariance Tento parametr udává maximální procentuální náhodný výkyv úhlů. Hodnota je typu `unsigned integer`.

DrawPipeCaps Povolí vykreslování podstavy na konci válcovitých těles. Hodnota je typu `unsigned integer`.

MinimizeTwist Povolí vyrovnávání příčného náklonu. Hodnota je typu `unsigned integer`.

SeparateGeometryForTranslucent Vynutí vytváření uzlu pro každé vzkreslené těleso s průhlednou texturou. Lze takto odstranit halo efekt u listů za cenu snížení výkonu. Hodnota je typu `unsigned integer`.

Color Nastaví barvu vykreslované geometrie. Při nastavení materiálů se barva neprojeví. Hodnota je vektor ve tvaru (`double, double, double`).

DiffuseMaterial Nastaví difuzní složku materiálu geometrie. Hodnota je vektor ve tvaru (`double, double, double`).

SpecularMaterial Nastaví spekulární složku materiálu geometrie. Hodnota je vektor ve tvaru (`double, double, double`).

AmbientMaterial Nastaví ambientní složku materiálu geometrie. Hodnota je vektor ve tvaru (`double, double, double`).

DiffuseTexture Nastaví texturu. Hodnota je cesta k souboru textury.

TextureSRepeatings Nastaví počet opakování textury po obvodu válcovité struktury. Hodnota je typu `unsigned integer`.

B.4 Tropismus

GravitropismElasticity Nastaví pružnost modelu vzhledem ke gravitropismu. Hodnota je typu `double`.

TropismElasticity Nastaví pružnost modelu vzhledem k diatropismu. Hodnota je typu `double`.

TropismVector Nastaví vektor pro diatropismus. Hodnota je vektor ve tvaru (`double, double, double`)

TropismAngle Nastaví úhel pro diatropismus. Hodnota je typu `double`.

B.5 Geometrie pro ladění

DrawDebugGeometry Povolí vykreslování geometrie pro ladění. Standardně vykresluje polohové vektory želvy při každém kroku. Hodnota je typu `unsigned integer`.

DebugGeometryScale Měřítko geometrie pro ladění. Hodnota je typu `double`.

Příloha C

Přiložené CD

K této diplomové práci je přiložen kompaktní disk s následujícím obsahem:

- Text této diplomové práce ve formátech PDF, PS a XML.
- Veškeré použité obrázky ve formátu JPG.
- Zdrojové kódy zásuvného modulu LSystem včetně projektových souborů pro Visual Studio 2005.
- Dokumentace k zásuvnému modulu LSystem.
- Zkompilovaný zásuvný modul LSystem ve formě dynamické knihovny DLL.
- Zkompilovaná aplikace Vrecko.
- Dávkové soubory s ukázkami generování stromů pomocí zásuvného modulu LSystem v prostředí Vrecko.
- Archívy použitých knihoven Boost a FunctionParser.