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 implementaci. 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

Kapitola 1

Úvod

Kapitola 2

Lindenmayerovy systémy

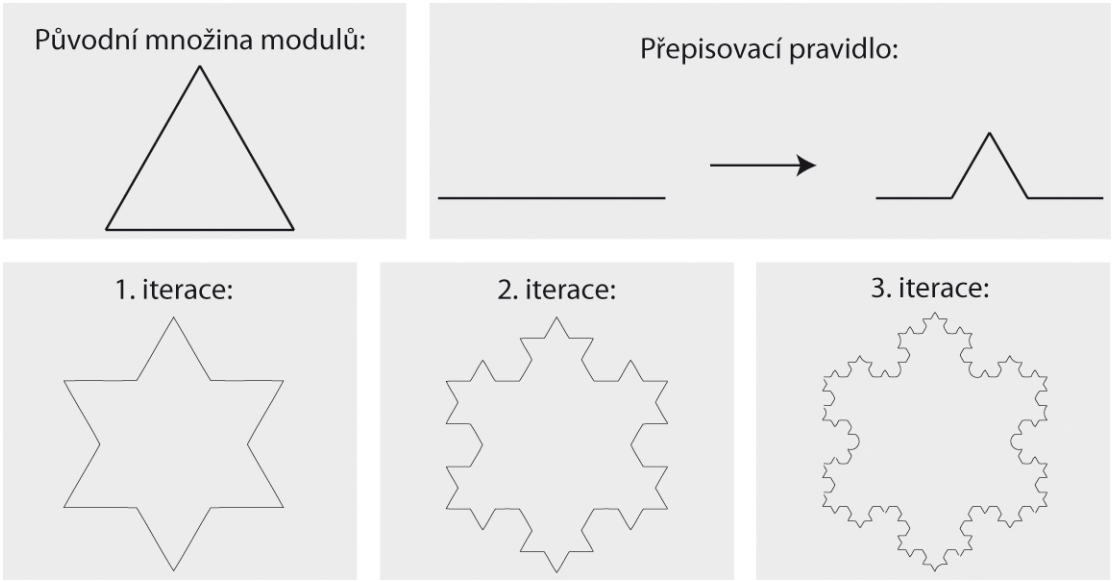
Lindenmayerovy systémy, zkráceně L-systémy, byly vytvořeny jako matematická teorie pro generování rostlin [4]. Původně byl kladen důraz spíše na obecnou topologii. Byly totiž vytvořeny pro simulaci vývoje větších částí rostlin nebo buněk mnohobuněčných organismů. Geometrie, ani podrobnější detaily v 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 použito modelování pomocí želví grafiky.

2.1 Vývoj rostlin pomocí procesu přepisování

Jádro L-systémů spočívá v použití přepisovacího systému. Jedná se o opakované nahrazování modulů předchůdců pomocí sady přepisovacích pravidel jejich následovníky, přičemž všechny moduly náleží do konečné abecedy modulů. Lze tak z jednoduchého původního objektu vytvořit opakováním přepisovacího procesu komplexní model. Modulem je myšlena libovolná atomická část modelu, jež většinou reprezentuje nějakou část simulovaného modelu. Pří 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 [3]. Šlo o jednoduchý bezkontextový přepisovací systém, jež obsahoval pouze jediné pravidlo, přepisující jeden grafický prvek na jiný (2.1). Později tento model podstatně rozšířil 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 odlišností L-systémů od Chomského gramatik spočívá v použití přepisovacích pravidel. Zatímco v Chomského gramatikách jsou přepisovací pravidla aplikována postupně, v L-systémech jsou použity paralelně v jednom derivačním kroku na všechny symboly přepisovaného řetězce. Motivací pro tento přístup byla podobnost s biologickými procesy. Příkladem mohou být mnohobuněčné organismy, ve kterých se dělí buňky současně. Na rozdíl od přírodních pochodů vývoj L-systémů probíhá v diskrétních krocích. Kvůli reprezentaci

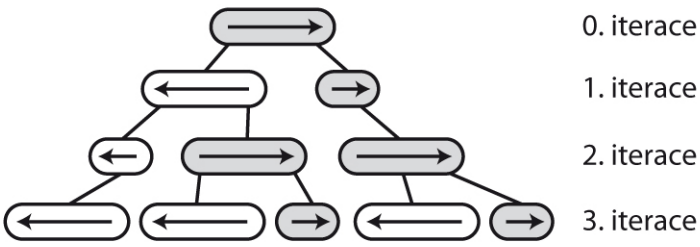


Obrázek 2.1: Kochova vločka a její první tři iterace přepisovacího procesu.

modelů jako řetězce znaků se často místo pojmu modul používá pojem 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-systém 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, jež se nachází v bakterii Anabaena catenula [5]. Tyto řetězce jsou tvořeny dvěma typy buněk. První typ jsou mladé, kratší buňky. Druhý typ jsou buňky starší a delší. Každá buňka má také svou polaritu, která určuje, kterým směrem bude buňka růst.



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

Formální definice D0L-systémů je následující [8]:

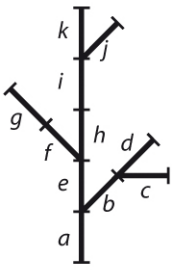
- Abeceda V je konečná množina *symbolů*.
- *Slovo* je posloupnost symbolů nad abecedou *V*. Množina všech těchto posloupností se označuje jako *V**.
- *Přepisovací pravidlo* je uspořádaná dvojice (a, u) zapsaná jako $a \rightarrow u$, kde a je symbol náležící *V** a u je slovo náležící *V**. Symbol a se nazývá *předchůdce* a slovo u *následník*.
- *D0L-systém* je 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 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.

2.3 L-systémy se závorkami

Obecné D0L-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 topologií je potřeba zavést dva moduly. Tyto moduly byly již součástí původního Lindenmayerova konceptu [4]. Pro označení začátku a konce větve se používají symboly pravé a levé hranaté závorky.



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

Slova, vyprodukovaná derivacemi přepisovacích pravidel závorkového L-systému, se označují jako stromy. Příklad takovéto stromové struktury vytvořené L-systémem se závor-

kami 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 hrotem hlavní osy. Obdobně lze pojmenovat i moduly na nižších úrovních a označit tak například moduly c , d , g a j jako hroty vedlejších větví.

Strom lze podle [7] definovat jako slovo w nad abecedou $V_E = V \cup \{[,]\}$ pro které platí, že:

- $w = x_1[a_1]x_2[a_2]...x_n[a_n]x_{n+1}$
- Podslova $x_1, x_2, ..., x_{n+1} \in V^*$ neobsahují moduly závorek.
- Podslova $x_1, x_2, ..., x_{n+1} \in V_E^*$ 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 simulovat 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 o překážce v růstu. Mozna dat tri typy pouziti – kapitola 7.3 handbook

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 < *předchůdce* → *následník* nebo *předchůdce* > *pk* → *následník*,

kde *lk* je levý a *pk* pravý kontext. Následující L-systém ukazuje, jakým způsobem je možné propagovat signál.

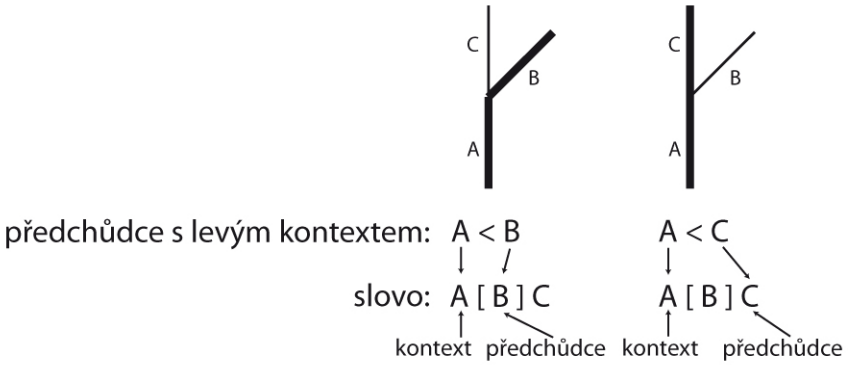
Obecnější třídu tvoří (k,l)-systémy, kde levý kontext tvoří slovo o délce k modulů a pravý kontext slovo o délce l modulů. V některé litaratuře bývají označovány ajko IL-systémy. V této práci se zabývám jejich podmnožinou, třídou 2L-systémů. Přepisovací pravidla v tomto případě zjišťují levý i pravý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

lk < *předchůdce* > *pk* → *následovník*.

U bezzávorkových L-systémů je proces zjišťování, zda přepisovací pravidlo odpovídá kontextu předchůdce, triviální a jednoznačné. Pro závorkové systémy se však literatura rozchází a jsou použity rozdílné podmínky. Jde především o zpracování pravého kontextu. Pro tuto práci jsem se rozhodl použít způsob, jež obdobně používá ve své práci James Scott Hannan [2]. 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. Tento způsob párování kontextů je tedy optimální pro použití při interakcích mezi jednotlivými moduly a mezi moduly a prostředím.

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, která 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 nesouvisí.

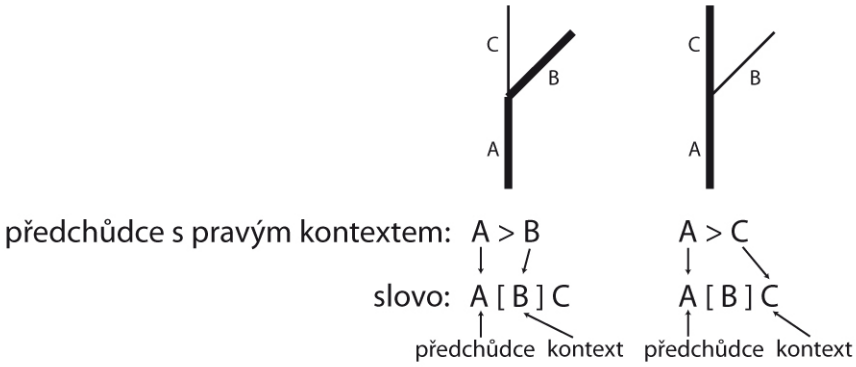


Obrázek 2.4: Párování levého kontextu

2.4.2 Párování pravého kontextu

Párování pravého kontextu srovnává 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 označeno jako odpovídající danému modulu předchůdce. Na obrázku 2.5 je znázoněno jednoduché větvení, kdy ke shodě dochází v případě, že pravý kontext bude modul *B* nebo i *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 $a_1, a_2, \ldots, a_n$ náleží do množiny $\Re$. Takovýto rozšířený modul se značí jako $A(a_1, a_2, ..., a_n)$ a patří do množiny $M = V \times \Re^*$, kde $\Re^*$ je množina všech konečných posloupností parametrů.

Pravidla parametrických 0L-systémů mají tvar

předchůdce (parametry) : podmínka $\rightarrow$ *následník*.

Parametry modulů následníka nemusí mít nutně tvar reálného čísla. Typicky jde o aritmetické výrazy, jež 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(*Σ*)*. Dalším rozšířením parametrických systémů je podmínka. Přepisovací pravidlo může být použito pro přepis pouze tehdy pokud je podmínka

splněna. Jedná se také o logický výraz označovaný jako *C(Σ)*. 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 z množiny.

Parametrické 0L-systémy [6] jsou definovány jako uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde

- *V* je konečná neprázdná *abeceda*,
- Σ je množina *formálních parametrů*,
- $\omega \in (V \times \Re)^+$ 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*.

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 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 liší mezi sebou. 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.

Takovýto parametrický stochastický 0L-systém lze definovat jako 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 se zde definuje funkce π , jež 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, jež na základě pravděpodobnostního faktoru určí pravidlo pro přepis. Tento faktor nenabývá obecně konstantních hodnot pro celý přepisovací proces, ale vyhodnocuje se, podobně jako podmínka a následník, na základě formálních parametrů předchůdce při každé iteraci. Pokud označíme $\widehat{P} \subseteq P$ jako množinu všech odpovídajících pravidel jednomu modulu, pak pravděpodobnost $prob(p_k)$ jednoho pravidla $p_k \in \widehat{P}$ odpovídá výsledku rovnice 2.6.1.

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

Rovnice 2.6.1: Výpočet pravděpodobnosti výběru pravidla p_k

Kapitola 3

Modelování rostlinných organismů

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 Przemyslaw Prusinkiewicz s myšlenkou interpretovat L-systémy jako pohyb želvy známé z programovacího jazyka LOGO [1]. Tato želva funguje jako kurzor, který přijímá různé rozkazy týkající se jeho pohybu. Na základě těchto rozkazů pak tento kurzor kreslí svým pohybem jednu spojitou čáru(???). Použití želví grafiky značně rozšiřuje geometrické možnosti L-systémů a je ideální pro tvorbu biologických struktur.

3.1.1 Planární želví grafika

Neboť 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 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.

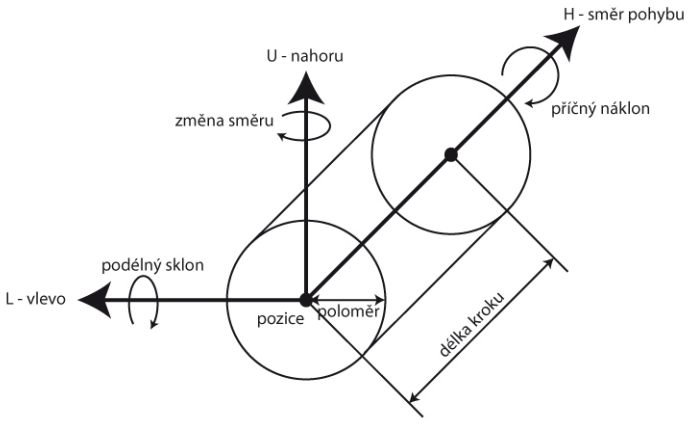
Při generování grafiky pak želva s počátečními parametry (x_0, y_0, α_0) interpretuje jednotlivé znaky řetězce postupně zleva doprava.

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 se se definuje vektor *P*. Pro určení orientace pak tři jednotkové vektory *H*, *L*, *U* takové, že splňují rozvici $H \times L = U$. Vektor *H* pododbně jako u rovinné želvy udává kam želva směřuje, tedy čelo. Vektor *U* směřuje směrem nahoru a vektor *L* pak směřuje vlevo od želvy.



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

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: směr, podélný sklon a příčný náklon. Kompletní sada příkazů, kterou intepretuje

i zásuvný modul, jež byl vytvořen v rámci této práce, je i spolu s popisem iterpretovaných funkcí uvedena v příloze LINK.

3.2 Subapikální L-systémy

Závorkové L-systémy umožnují 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 nazývané 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 Kelemenovou v roce 1987(7 apical). Myšlenkou těchto L-systémů je, že k větvení dochází pouze u vrcholů již exitujících větví. Tato vlastnost výchází ze základních výsledků při pozorování rostoucích rostlin. Nové rostlinné orgány, jako je stonek, větve, listy nebo květy mohou být vytvořeny pouze z apikálního meristému. Tato tkáň obsahuje aktivně se dělicí buňky a nachází se v oblasti vrcholů větví.

V přírodě rozeznáváme tři typy vzorů větvení jež splňují podmínku subapikálních L-systémů. Jednoduché bazitonické a složitější mezotonické a akrotonické vzory.

3.2.1 Bazitonický vzor větvení

Jedná se o struktury v nichž je větvení u vrcholu rostliny méně vyvinuté než větve blíže základny rostliny. Bazitonické struktury lze vytvořit i za pomocí jednoduchých D0L-systémů. Klasic

3.2.2 Mezotonický a Akrotonický vzor větvení

Kapitola 4

Zásuvný modul LSystem

Stěžejní části této diplomové práce je návrh a implemtace zásuvného modulu pro systém Vrecko. Tento modul umožnuje simulaci rostlin, stromů ale i jiných struktur 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 funkcionality.

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 derivačních kroků L-systémů. Slova i ostatní data jsou během derivačního procesu i poté uložena do datových struktur optimalizovaných pro daný účel. Jelikož existuje několik typů L-systémů, je implementována funckionalita, která pro přepisování vybere algoritmus, který je svými schopnostmi i časovou náročností nejvhodnější pro zpracovávaný L-systém. Tvorba L-systémů se složitější strukturou se většinou neobejde bez hiearchického 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 rozdílný L-systémy pro listy, květy a plody a následně je vložit do L-systému kostry stromu. Podrobněji se touto části funcionality zabývá kapitola ???.

Druhý celek se zabývá grafickou interpretací vygenerovaného řetězce. Komunikace s prvně zmiňovaným celkem je 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 i interpret, jež k vytváření geometrie neslouží. Jedná se o intepret pro zpracování dotazů během generování iterací L-systémů. Je tedy optimalizován pro rychlé zjištění polohy a orientace želvy. Pro zpracování slov obsahující závorky má interpret implementován zásobník. Uživatel má také možnost zvolit si z několika typů želv, jež pak dovolují vykreslovat geometrii rozdílnými přístupy. Poslední celek plní funkci řídícího modulu a je umístněn nad oběmi předcházejícími celky, jež spravuje. Umožňuje také nahrání a ukládání vygenerovaných slov. Odpadá tak opětovné generování při jejich dalším použití. 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 2010 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. Mimoto byly pro vývoj použity některé z knihoven Boost. Jednak jde o knihovnu Program properties pro jednoduché a intuitivní ukládání a zpracovaní nastavení a konfiguračních souborů a dále pak o knihovnu Lexical cast, která jednoduše převádí a přetypovává řetězce znaků. 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 obejktově orientované rozhraní nad OpenGL. Nízkoúrovňová funkcionalita je zde převedena na objekty v grafech scén. Zároveň jsou možnosti OpenSceneGraphu oproti klasickému OpenGL značně rozšířené a je přidáno mnoho funkcionalit, jež zjednodušují a urychlují práci programátora.

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 demonstrují 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 jež mají určité vlastnosti a dovednosti. Díky ním lze objekty ovládat a měnit jejich tvar i vlastnosti. Zároveň lze k tomuto systému připojit řadu různých zařízení pro interakci obsahu scény s uživatelem.

Kapitola 5

Generovaní 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 rychlost a také na modularitu. Rychlost je zde důležitá zejména při iteračním procesu, neboť výsledné slovo může mít délku až v řádu miliónů modulů. Je tedy třeba zajistit vhodné struktury pro ukládání dat a vybrat vhodný algoritmus pro rychlý přepis pravidel. Modularita a rozšířitelnost zde s rychlostí 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 dýležitá i pro syntaktický analyzátor jež tak zvládne zpracovat více formátů L-systémů.

Celou tuto funkcionalitu a komunikaci s okolím zajišťuje třída LSystemGenerator. Tato třída je potomkem třídy AbstractGenerator jež 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 nejvhodnější. 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 intepretace. Zásuvný modul nyní podporuje dva typy souborů a lze jej rozšířit o podporu dalších formátů. Jednat se jedná o mírně pozměněný textový formát LS, jež v obdobné formě používá ve svých projektech výzkumná skupina Biologického Modelování a Vizualizace kolem prof. Przemyslawa 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, jež byl zvolen pro svou jednoduchou rozšířitelnost a přehlednost.

Struktura je v obou případech hiearchická a a popis jejich entit je následující.

- **Unikátní identifikátor L-systému** je nutné přiřadit každému L-systému. Ten se pak používá hlavně při vkládání subsytémů nebo pro získávání parametrů o daném L-systému.
- **Typ L-systému** je důležitým a nutným parametrem L-systému. Určuje základní vlastnosti jako je determiničnost a bezkontextovost L-systému. Na jeho správném nastavení pak záleží při výběru 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 i interpretaci slova globální parametry získané z konfiguračního souboru. Kompletní popis jednotlivých parametrů se nachází v příloze A.
- **Subsystémy** lze vkládát do každého L-systému. Lze je vložit přidáním cesty k souboru L-systému. Nikdy však není vhodné tvořit v grafu hiarchie 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** jsou množina, která budou použita při iteračním procesu. Na pořadí není brán zřetel, protože se při každém iteračním kroku aplikují všechny pravidla najednou. Z tohoto důvodu je však důležité, aby byla jednotlivá pravidla jednoznačná a neexistovaly dvě pravidla pro stejný modul předchůdce. Jedinou výjímkou 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, jež 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_Subsytems;
    void substitute(std::map<std::string, std::string> & pairs)
public:
    AbstractFile();
    virtual void open( std::string & ) = 0;
    virtual std::vector<string> * getHomomorphisms()
    virtual std::vector<string> * getRules();
    virtual std::vector<string> * getSubsystems();
```

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

Třídy odvozené od třídy `AbstractFile` musí implementovat metodu `open()`, jež zaručí zpracování dat ze souboru a uloží je do jednotlivých atributů této třídy. Metoda `substitute()` slouží jako výpomocná metoda při nahrazování řetězců a může být použita pro zpracování konstant nebo subsystémů. Ostatní metody slouží jen přístupové metody k atributům.

5.1.1 Formát LS

Jedná se o jednoduchý textový formát založený na příkazech jež jsou podobné 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 LeafPitch 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)[^(LeafPitch)#{Leaf}][#{Leaf}]Z[^(LeafPitch)#{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. Kořenovou značkou je `LSystem` a ta pak 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 je odpovídajícími XML entitami.

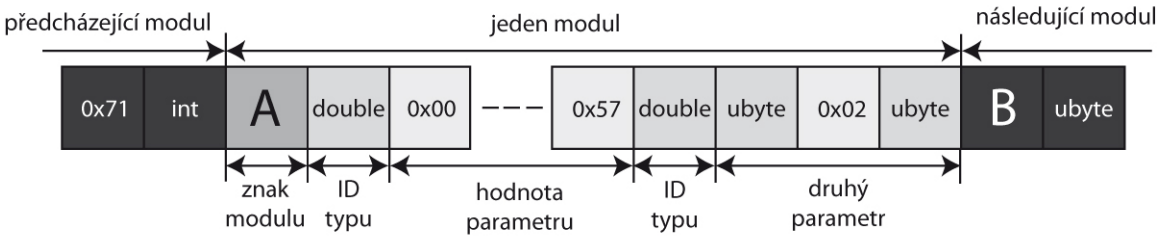
5.2 Řetězce modulů

Při generování slova L-systémem dochází k vytváření velmi dlouhých řetězců s nestejnorodými 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 proto vytvořena na míru přizpůsobená třída `LongString`. Data jsou zde uložena ve formátu, který umožňuje rychlou manipulaci s daty,

především pak připojení nového řetězce na konec stávajícího. Rychlost provedení této operace je potřeba při iteračním procesu. Ten při každém kroku zpracovává původní slovo a vytváří podle něj na základě přepisovacích pravidel nové slovo další iterace. Provádí to tak, že nahlíží na 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 je 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 dvěma bajty zleva a 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. Současná implementace umožňuje pouze použití modulů, které mají všechny parametry shodného typu.

5.2.2 Inicializace

Pro vytvoření instance slouží jediný konstruktor. Jediným 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 kapi-

tole, 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. Příklad takovéhoto řetězce:

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 idetifikátoru odpovídá pořadí ve výčtu ParameterType.

5.2.3 Připojování řetězců

Jak již bylo výše zmíněno, je tato struktura optimalizována pro opakované 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 Připojování řetězců

Přistupovat lze k datům jedním ze čtyř způsobů.

- Pomocí operátoru `[]` získat přímo 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.
- Pro zpracování parametrických L-systémů se používá jedna z metod šablony `getParameters()`, která vrací pole parametrů libovolného typu a jejich počet. Tyto parametry jsou získány z pozice za identifikujícím znakem.
- Metody `matchRight()` a `matchLeft()` slouží především k ověřování kontextu. Pro svou funkci využívají metody `peekSymbol()` pro nahlédnutí na levý nebo pravý nejbližší znak a jelikož řetězce obsahují i závorky, jsou implementovány i pomocné metody `findMatchingRightBracket()` a `findMatchingLeftBracket()` pro nalezení odpovídajícího protějšku k nalezené 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 věnuje kapitola LINK. 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 zkomponovat 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 jejímž 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 dostupnou knihovnu, která poskytuje dostatečnou a navíc uživatelsky rozšiřitelnou funkcionalitu. Nabízí i možnosti optimalizace. Pro každý výraz, jež 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.

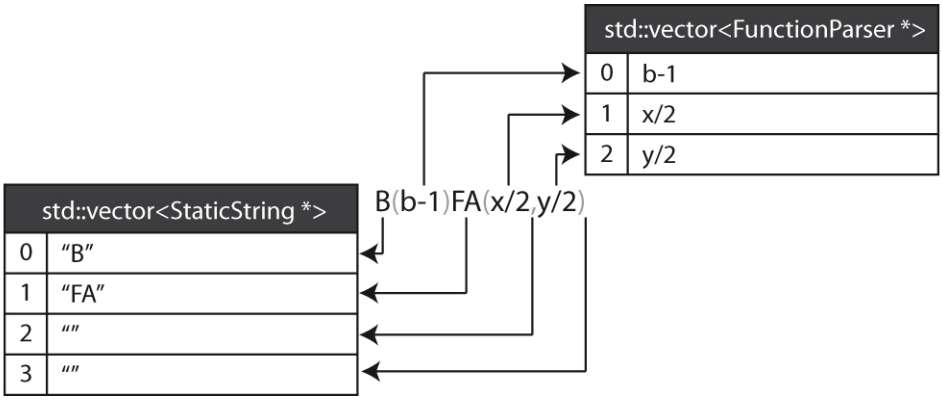
Jako příklad je zde uvedeno pravidlo stochastického parametrického 2L-systému, kterým se zabývala kapitola 5.5.3.

```
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 zřejmé, že pro všechny výrazy se používají pouze proměnné předchůdce a kontextů. V tomto případě se jedná o `x`, `y`, `z`, `a`, `b`, `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 odkazy 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 nelze zvětšit. Pro účel krátkých neměnných řetězců následníka je tedy ideální, protože nezabírá tolik prostoru v paměti. Řetězce i výrazy se ukládájí do konejneru

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: Rozdělení následníka na řetězce modulů a výrazy

Z příkladu 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 obsahují, vyskytuje se většinou někde uprostřed následníka. Proto je tento způsob zpracování optimálnější.

5.4 Zpracování L-systémů

Všechny tyto typy jsou implementovány jako potomci třídy `AbstractLSystem`. Obrázek LINK ukazuje hiearchii tříd L-systémů.

Abstraktní třída AbstractLSystem obsahuje několik metod, sloužící jako rozhraní pro L-systémy.

- Pro inicializaci slouží metoda `loadFromFile()`, která načte všechna potřebná nastavení. Jako zdroj nastavení slouží některá z instancí potomků abstraktní třídy `AbstractFile`.
- Pro provedení další iterace slouží metoda `nextIteration()`. Počet iterací lze ovlivnit i nastavení 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.

Veškerá funcionalita, 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 nejvhodnějšího 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 sybsystémů. 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 základě níž lze rozpoznat zda daná třída požadavkům vyhovuje. Používá k tomu statického atributu capabilities, neboť zde jsou schopnosti uloženy jako jejich bitový součet. Jednotlivých schopností jsou definovány ve výčtu LSystemCapabilities.

Vytvoření nejvhodnější instance usnadňuje statická tovární metoda `AbstractGenerator::createLS` jež 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 zde 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 `addHomomoprhism()` 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.zpracujPodmínku( 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 puvodniSlovo;
LongString noveSlovo;
for( všechny moduly v puvodniSlovo )
{
    najdi všechny pravidla, které mají aktuální modul jako předchůdce;
    if( existuje alespoň jedno takové)
    {
        if(vyber pravidlo)
        {
            noveSlovo.připoj( vygeneruj následníka modulu);
        }
        else
        {
            noveSlovo.připoj(použij pro daný modul pravidlo identity);
        }
    }
    else
    {
         noveSlovo.připoj(použij pro daný modul pravidlo identity);
    }
}
```

Prvotní prohledání pravidel vybere pouze ty, které 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 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 LINK. 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 LINK o typech větvených struktur, je nutné pro modelování pokročilejších rostlinných organismů použít složitější typy L-systémů. Zároveň však není kvůli optimalizaci vhodné použít stejných L-systémů pro generování jednoduchých struktur. Navíc mají různé typy L-systémů různé formy zápisů svých přepisovacích pravidel. V následující kapitole jsou popsána jednotlivá specifika implementace jednotlivých typů L-systémů a také formáty zápisů pravidel, jež tyto systémy zpracovávají. Jsou zde rozebrány pouze vlastnosti, které nebyly společné a tak byly v minulé kapitole LINK zmíněny jen okrajově.

5.5.1 D0L-systémy

Nejjednodušším L-systémem je tato varianta implementovaná třídou `D0LSystem`. Jak již název napovídá, zpracovává tato třída pouze deterministické bezkontextové L-systémy. Pro řadu modelů je tento typ generování dostatečný. Navíc je díky své jednoduchosti nejrychlejší. Zápis pravidel je ukázán na následujícím vzorovém příkladu.

```
A->A+(10)FBF
```

Tento typ tedy 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.

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 parametrům lze vytvořit i mezotonické a akrotonické struktury. Náhodnost pak dovoluje vytvářet rostlinné organismy s rozdílnou a náhodnout strukturou, jež však mají společné rodové znaky. Jsou implementovány ve třídě ParStoch0LSystem. Tato třída zpracovává L-systémy ve tvaru, jak je popisuje kapitola LINK. Příkladem mohou být následující pravidla zapsané již v podobě, jež je tímto L-systémem zpracovatelná.

```
A(x,y):(x>y)->A(x-1,y)FB(y/2)F:(x*y)
```

Jednotlivé části tohoto řetězce mají následující význam.

- A(x,y) označuje předchůdce pravidla se dvěmi parametry x a y. Tento údaj je povinný,
- (x>y) je příklad podmínky. Pokud pravidlo podmínku neobsahuje, může být podmínka nahrazena symbolem *, který značí, že je podmínka splněna vždy.
- Následník A(x-1,y)FB(y/2)F bude rozdělen 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. Tyto 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 podmínku splňuje 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 použito pro přepis.

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 LINK. Pro tuto činnost byly implementovány metody `matchRight()` a`matchLeft()`. Jde o metody třídy LongString a vrací hodnotu int, která informuje zda bylo ověřování úspěšné. V případě úspěchu vrací tyto metody 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 větvení, 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. Tyto parametry se načtou z řetězce slova díky pozicím modulů získaných během ověřování kontextu jako návratové hodnoty. Díky rozšíření o zpracování kontextu mohl být do zásuvném modulu implementován také mechanismus pro zpracování dotazů. Podrobnějším popisem implementace dotazů se zabývá následující kapitola 5.6.

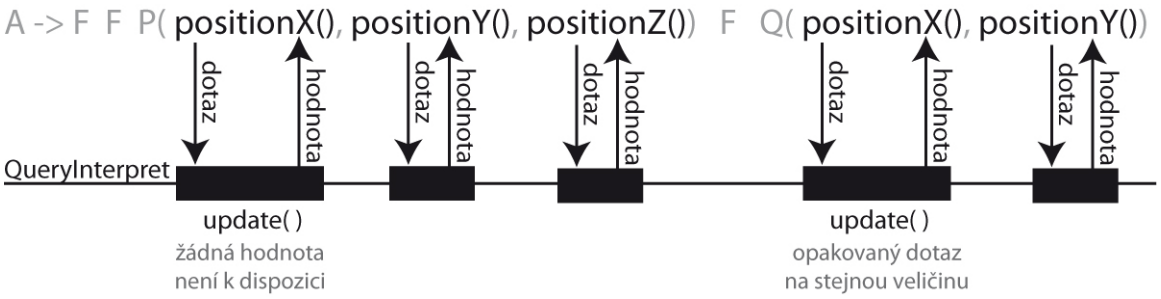
5.6 Dotazy

Díky nim lze již během generování zjistit, kde a v jaké poloze se bude daný modul nacházet v prostoru. Díky rozšířitelnosti knihovny `FunctionParser` mohly být přidány do výrazů funkce, jež kterýkoli parametr polohy želvy vrátí. 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 funkce jsou ve skutečnosti statickými metodami třídy `Query`. Každá tato metoda je bezparametrická a vrací hodnotu double. Pro každou souřadnici každého vektoru tedy existuje jedna metoda.

Třída `Query` však slouží pouze jako adaptér mezi třídami `FunctionParser` a `QueryInterpret`, což je speciálně uzpůsobený interpret pro zpracování dotazů. Jeho fungování je podobné jako u třídy pro intepretaci geometrie, jež bude popsána v kapitole 6.2. Narozdíl od ní však nevytváří žádnou geometrii a intepretuje 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 dotazovaní metody. Celá tato operace získání polohy je zajišťována 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. Je založen na principu, že pokud je některá souřadnice přečtena podruhé, jedná se o jiný modul a je tedy třeba provést obnovu. Na obrázku lze vidět, jakým způsobem vypadá pravidlo s dotazy a jak se jednotlivé dotazy pomocí instance třídy

`QueryInterpret` vyhodnocují. U dotazů, které si vynutí provedení interpretace slova je spolu s důvodem uvedena metoda `update()`. Ostatní dotazy obdrží odpověď okamžitě.



Obrázek 5.3: Zpracování dotazů třídou QueryInterpret

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

Zpracováním dotazů lze tedy do řetězce slova dostat poziční hodnoty ještě před fází vykreslování. Kombinací s kontextovými pravidly s pomínkou lze dosáhnout toho, že některá pravidla budou použita jen při urcčitých hodnotách těchto dotazů. Druhou možností je využít tyto hodnoty ke změně parametrů modulů pravidla. Například v případě popinavých 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 hiearchických úrovní. Pro lepší kontrolu, přehlednost a také nastavitelnost je lepší jednotlivé dílčí celky hiearchicky seřadit. U botanických organismů se může jednat například o rozdělení na rostlinné orgány. Jde však jí samozřejmě 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 tedy vytvořím 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 ponechána 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 intepret

o změně L-systému, který pak dle parametrů tohoto modulu přepne na zpracování řetězce dle odpovídajících nových parametrů. Neboť je toto chování dosti podobné zpracování závorkám, věnuje se jim společná kapitola o implementaci zásobníku LINK.

Kapitola 6

Generovaní slov pro modely rostlin

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 intepretují a vytváří tak jeho geometrickou podobu. Pro intepretaci 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 geomterie 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 již pouze o intepretaci, 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.

Pro interpretaci slov slouží abstraktní třída AbstractInterpret. Implementováni jsou dva potomci. TurtleInterpret zpracovává věškeré příkazy želvy a slouží k vytváření geometrie. QueryIntepret provádí zpracování dotazů ohledně polohy želvy. Tento interpret zpracovává jen vybrané příkazy. Podrobněji byl popsán v kapitole LINK.

Řetězec generovaný L-systémem je přetypován na nový typ. Z intstance třídy LongString je konvertován na instanci třídy ParseableString. Jedná se o jednoduchou třídu, která obsahuje metody pro dopředné procházení řetězcem a získávání parametrů.

6.1 Spojení s grafem scény

Pro zobrazení generované geometrie je důležité napojení na graf scény. Při vytváření interpretu je mu kontruktorem 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 pro generování geometrie 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 LINK.

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 buď již nastaveny nebo rychle připraveny ke zpracování.

6.2 Interpretace pomocí želví grafiky

Doplnit info o Interpretu a metode parse() Úkolem interpretu je projít celé slovo typu ParseableString a moduly, které rozpozná převést na příkazy pro želvy. Pro tento převod modulu na příkaz slouží metoda parse(). Každá z tříd

6.2.1 Zásobník pro ukládání želvích instancí

Pro zpracování slov, které obsahují závorky a podsystémy, je nutná dobrá implementace zásobníku pro želvy. K tomuto účelu slouží třída TurtleStack, která poskytuje standardní funkce zásobníku. Je však uzpůsobena pro použití v tomto zásuvném modulu. Při procházení řetězce a interpretaci jednotlivých modulů se operace prov K operacím na zásobníku dochází v případě některých speciálních symbolů.

-] — Tato závorka označuje počátek nové větve. Je tedy nutné vytvořit na zásobníku metodou push() novou želvu, která se vykreslováním této větve bude zabývat. Parametry i typ si tato želva nastaví podle želvy na vrcholu zásobníku.
-] — Ukončení větve je signalizováno tímto symbolem. Želva je z vrcholu zásobníku odstraněna a v generování dále pokračuje želva pod ní.
- $(x) — Jednotlivé L-systémy mají již od svého načtení přiřazen jednoznačný identifikátor. Symbol dolaru s jedním parametrem oznamuje intepreteru, že má dojiít ke změně L-systému. Podle identifikátoru v parametru modulu se zvolí správný L-systém a jeho odpovídající uzel s geometrií LSGeode. Ten už poté ví, který typ želvy použít a jaké ji nastavit výchozí parametry. Některé parametry, jako je matice polohy, mohou být nastaveny podle želvy na vrcholu zásobníku. S novou 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 tak pokračuje želva nadřazeného L-systému.

6.2.2 Želví rozhraní

Pro větší modularitu byl zvolen při návrhu model, jež umožňuje použití libovolného množství typů želv. Lze tedy generovat různou geometrii pomocí stejných mechanismů. Všechny třídy želv dědí z abstraktní třídy AbstractTurtle. Tato třída obsahuje rozhraní pro úplnou

kontrolu želvy. Většinu tvoří metody pro zpracování všech příkázů, které se ve slovech vyskytují jako moduly. Kromě toho obsahuje metody pro 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. Tyto parametry si želvy mezi sebou předávají při operacích na zásobníku. Jinou možností nahrání výchozích parametrů z instance třídy LSGeode.

6.2.3 Želví příkazy

Každá želva intepretuje celou řadu příkazů. Ne všechny želvy musí nutně implementovat všechny příkazy. Některé mohou být velice úzce specializované a mít implementováno jen několik příkazů. Obecně však 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, vyrovnání náklonu do vodorovné polohy nebo náhodné otočení v libovolném směru.

- Změna stavových proměnných — Řadu parametrů želvy lze během intepretace měnit. Je tak možné například prodloužit délku kroku nebo měnit výchozí úhel rotace.

- Nastavení vnějších vlivů — Některé příkazy mohou ovlivnit reakci dané 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 v 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 kontejnery s geometrií. Díky tomuto propojení uzlů 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 krajními body kroku želvy. Protože by při rotaci mezi jednotlivým 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`.

6.3.2 Spojité válce

Želvy tohoto typu generují podobně jako předchozí typ geometrii složenou z válců. Díky odlišnému přístupu zde však 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 kontúry, která pak tvoří horní podstavu předešlé válcovité geometrie a zároveň spodní podstavu geometrie kroku, který bude případně následovat. Rovina kontúry 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, jež navíc kvůli absenci polokoulí obsahuje menší množství trojúhelníků. Mnohem lépe jde zde také vyřešit navazování textur.

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 nanášejí 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. 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 způspob nanášení textury lze nalézt v této práci LINK. Implementace v tomto zásuvném modulu ale navíc umožňuje opakování ve směru s. Způsob generování texturových souřadnic je znázorněn na následujícím obrázku. Celý válec představuje jeden krok želvy. Kolem obvodu se textura na tomto obrázku opakuje dvakrát. Podélná textura je pak adekvátně přizpůsobena.

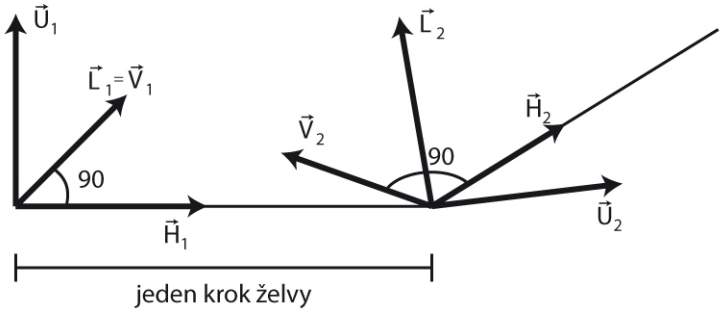
6.4.2 Mapování textur na obdélníky

Pro nanášení textury na objekty listů je použita jednoduchá metoda mapování. Neboť 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* pro správné zobrazení průhledných objektů obsahuje speciální koš. Geometrie typu `osg::Geometry` jsou pak v tomto koši vykreslovány postupně odzadu dopředu tak, aby byly i s průhledností vykresleny korektně. Kvůli optimalizaci je však vhodné mít všechny objekty listů v jedné geometrii, což 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 `SeparateGeometryForTranslucent`, který vynutí použití oddělených geometrií pro každý objekt listu. Listy jsou v tomto módu korektně seřazeny. Dochází však vinou velkého množství geometrií ke snížení výkonu.

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. Jedno z nich je i deformace válců při provádění rotace kolem osy H. V každém kroku želva generuje body válcovité geometrie podle své kontury a řídí se aktuální orientací želvy a tedy jejího vektoru L. Pokud v jednom kroku dojde k příčnému náklonu o příliš velký úhel, může dojít k deformaci geometrie. LINK OBRAZEK. Někdy může být tento efekt žádoucí, většinou je ale lepší jeho odstranění. Řešením může být jeden z několika algoritmů pro minimalizaci příčného náklonu. Pro zásuvný modul LSystem byla zvolena Bloomenthalova metoda minimalizace rotace, kterou Bloomethal obdobně použil při generování svého modelu "Mohutný javor". Kromě polohové matice želvy se zavádí navíc matice kontúry. Tato matice pak slouží ke generování bodů kontúry a vytvoření válcovité geometrie, která nepodléhá příčnému náklonu. Tato matice je vytvořena na základě FRAME vektorového prostoru $(H, V, H \times V)$, kde H je směrový vektor želvy a V je vektor, podle něhož se bude generovat geometrie. Tento vektor je před interpretací slova nastaven jako shodný s vektorem L.

Při každém kroku želvy kupředu jsou v metodě `adjustMatrices()` hodnoty vektorů upraveny. Pokud označíme H_1 jako směrový vektor želvy původního FRAME a H_2 označíme směrový vektor želvy po vykonaném kroku vpřed, můžeme nový FRAME získat oto-



Obrázek 6.1: Princip minimalize příčného náklonu u dvou následujících FRAMů želvy

čením původního FRAME kolem osy $H_1 \times H_2$ o úhel, který svírají vektory H_1 a H_2 . Úhel V pak bude vždy ve stejné rovině s úhlem L. Narozdíl od něj však 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 modely podstatně přiblížit realitě. Pro směrové podněty se používá termín tropismus. Jedná se o změnu orientace rostlinného organismu na základě nějakého podnětu či podráždění LINK PLANT FORM. 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 směřování 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 $\overrightarrow{H}$ směrem k vektoru podnětu $\overrightarrow{T}$.

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

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

Při každém kroku želvy je metodou `adjustMatrices()` vypočten pomocí výše zmiňovaného 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 nastavení úhlu tropismu. Ohebnost se zde nastavuje parametrem `TropismElasticity`.

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

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

Rovnice 6.6.2: Výpočet korekčního úhlu při diatropismu.

6.7 Odezva na směrové podněty

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 již vygenerovaného slova vytvářet ne zcela identické jedince. Druhým argumentem pro zavedení toho mechanismu je často absolutní matematická přesnost úhlů generovaných rost-

lin. Mírným náhodným roztřesením úhlů lze odstranit viditelnou pravidelnost a dosáhnout tak mnohem lepších vizuálních výsledků.

V zásuvném modulu LSystem lze tohoto efektu dosáhnout nastavením parametru `AngleVariance`. Jde o procentuální hodnotu, která udává o kolik procent se mohou úhly maximálně změnit. Při provádění libovolné rotace želvy kolem jedné z jejich os, je pak úhel otočení upraven náhodnou hodnotou získanou na základě tohoto parametru.

Literatura

[1] Abelson, H. a DiSessa, A.: *Turtle geometry*, M.I.T. Press, 1982. 3.1

[2] Hanan, J.: *Parametric L-systems and their application to the modelling and visualisation of plants*, , 1992. 2.4

[3] von Koch, H.: *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes*, Acta mathematica, 1905. 2.1

[4] Lindenmayer, A.: *Mathematical models for cellular interaction in development, Díl I a II*, Journal of Theoretical Biology, 1968. 2, 2.3

[5] Mitchinson, G. a Wilcox, M.: *Rules governing cell division in Anabaena*, Nature, 1972. 2.2

[6] Prusinkiewicz, P. a Lindenmayer, A.: *The Algorithmic Beauty of Plants*, Springer-Verlag, 1990, 9780387972978. 2.5

[7] 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

[8] Rozenberg, G. a Saloma, A.: *The mathematical theory of L-systems*, Academic Press, 1980. 2.2

[9] 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

Při interpretaci se želvy řídí moduly, jež obsahuje zpracovávané slovo. Tato příloha obsahuje přehled všech intepretovaných modulů, jejich identifikující znak a popis příkazu, který vykonají.

A.1 Oddíl