

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

1	Úvod . . . . .	1
2	Lindenmayerovy systémy . . . . .	2
2.1	Vývoj rostlin pomocí procesu přepisování . . . . .	2
2.2	D0L-systémy . . . . .	3
2.3	L-systémy se závorkami . . . . .	4
2.4	Parametrické L-systémy . . . . .	4
3	Modelování rostlinných organismů . . . . .	6
3.1	Interpretace želví grafikou . . . . .	6
3.1.1	Planární želví grafika . . . . .	6
3.1.2	Interpretace závorkových L-systémů . . . . .	7
3.1.3	Rozšíření pro interpretaci L-systémů v 3D . . . . .	7
3.2	Subapikální L-systémy . . . . .	7
3.2.1	Bazitonický vzor větvení . . . . .	8
3.2.2	Mezotonický a Akrotonický vzor větvení . . . . .	8
4	Zásuvný modul LSystem . . . . .	9
4.1	Vývoj zásuvného modulu . . . . .	10
4.2	OpenSceneGraph . . . . .	10
4.3	Vrecko . . . . .	10
5	Generování slov pro modely rostlin . . . . .	11
5.1	Soubory L-systémů . . . . .	11
5.1.1	Formát LS . . . . .	13
5.1.2	Formát XML . . . . .	13
5.2	Soubory L-systémů . . . . .	14
5.2.1	Vnitřní struktura řetězce . . . . .	14
5.2.2	Inicializace . . . . .	15
5.2.3	Připojování řetězců . . . . .	15
5.2.4	Připojování řetězců . . . . .	15
5.3	Implementace přepisovacích pravidel . . . . .	16
5.4	Zpracování L-systémů . . . . .	17
5.4.1	Výběr nejvhodnějšího L-systému . . . . .	17
5.4.2	Inicializační fáze . . . . .	18
5.4.3	Přepisovací fáze . . . . .	18
5.5	Rozdělení L-systémů . . . . .	19
5.5.1	D0L-systémy . . . . .	19
5.5.2	Parametrické stochastické bezkontextové L-systémy . . . . .	20
5.5.3	Parametrické stochastické kontextové L-systémy . . . . .	20
5.6	Dotazy . . . . .	22
5.7	Podřízené L-systémy . . . . .	22
6	Generování slov pro modely rostlin . . . . .	24
6.1	Spojení s grafem scény . . . . .	24

6.2	<i>Interpretace pomocí želví grafiky</i>	25
6.2.1	Zásobník pro ukládání želvích instancí	25
6.2.2	Želví rozhraní	25
6.2.3	Želví příkazy	26
Literatura		27
A	<b>Příloha</b>	28

## **Kapitola 1**

### **Úvod**



## Kapitola 2

### Lindenmayerovy systémy

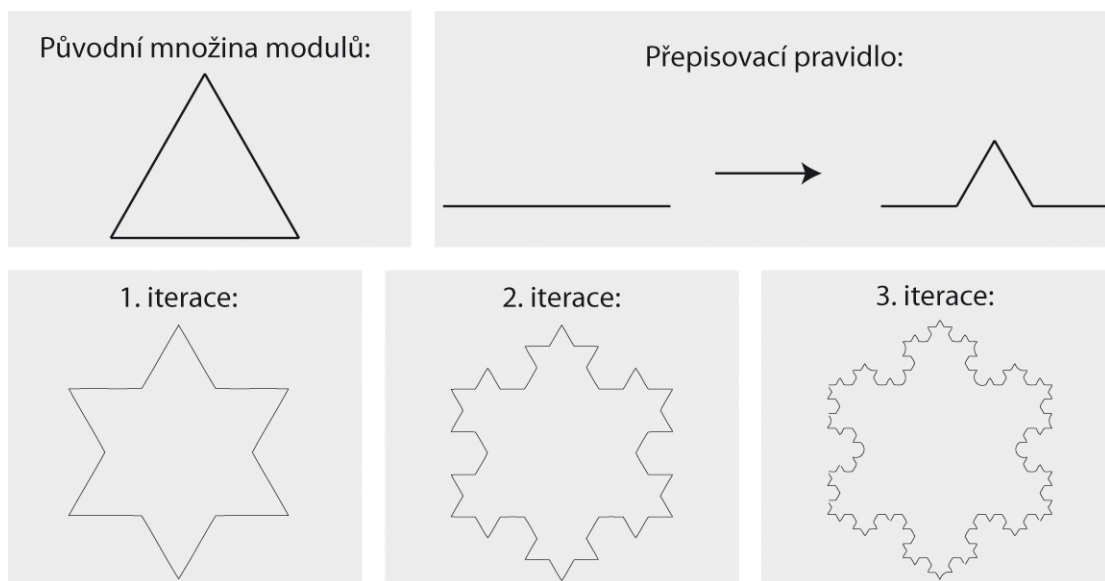
Lindenmayerovy systémy, zkráceně L-systémy, byly vytvořeny jako matematická teorie pro generování rostlin [2]. Původně byl kladen důraz spíše na obecnou topologii. Byly totiž vytvořeny pro simulaci vývoje větších částí rostlin nebo buněk mnohobuněčných organismů. Geometrie, ani podrobnější detaily v této původní teorii zahrnuté 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. 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 [1]. 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

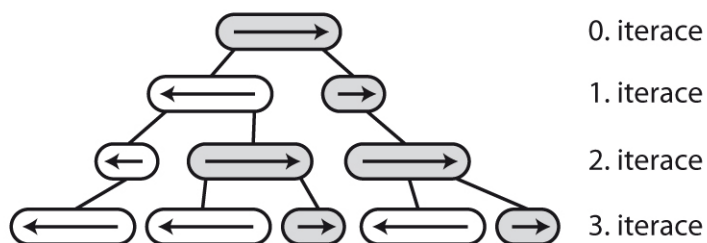


Obrázek 2.1: Kochova vložka a její první tři iterace přepisovacího procesu.

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 (str 5 ABOP) je znázorněna simulace vývoje mnohobuněčného vlákna buněk, jež se nachází v bakterii *Anabaena catenula* [3]. 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 [5]:

- 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  $\epsilon$  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 [2]. Pro označení začátku a konce větve se používají symboly pravé a levé hranaté závorky.

Slova, vyprodukovaná derivacemi přepisovacích pravidel závorkového L-systému, se označují jako stromy. Příklad takovéto stromové struktury vytvořené L-systémem se závorkami je na obrázku (str 9 fig 3.2 handbook). Jedná se o základní větvenou strukturu s jednou hlavní osou tvořenou moduly *aehik*. Slovo reprezentující tento strom lze pomocí jednoho řetězce zapsat následujícím způsobem:

$$w = a [ b [ c ] d ] e [ f g ] h i [ j ] k$$

Strom je dle (98) slovo  $w$  nad abecedou  $VE = V \cup \{[, ]\}$ , kde  $V$  je abeceda všech modulů z nichž se skládají větve. Větev je pak libovolné slovo generované L-systémem bez závorek.

### 2.4 Parametrické 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. Možná tři typy použití – 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 „ $lk < \text{předchůdce} \rightarrow \text{následník}$ “ nebo „ $\text{předchůdce} > pk \rightarrow \text{následník}$ “, kde  $lk$  je levý a  $pk$  pravý kontext. Následující L-systém ukazuje, jakým způsobem je možné propagovat signál.

## 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 obrazy. 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 ABOP). 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  $(x, y, \alpha)$ , kde  $x$  a  $y$  udávají kartézské souřadnice želvy a úhel  $\alpha$  směr, kterým želva míří, tzv. čelo. 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.
- **+** — Prikáže želvě otočit se o předem definovaný přírůstek úhlu doleva. Úhel želvy se zvětší o tuto hodnotu.
- **-** — Prikáže želvě otočit se o předem definovaný přírůstek úhlu doprava. Úhel želvy se zmenší o tuto hodnotu.

Při generování grafiky pak želva s počátečními parametry  $(x_0, y_0, \alpha_0)$  interpretuje jednotlivé znaky řetězce postupně zleva doprava.

### 3.1.2 Interpretace závorkových L-systémů

Jak již bylo výše zmíněno, pro simulaci větvičích 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 definuje vektor  $P$ . Pro určení orientace pak tři jednotkové vektory  $H, L, U$  takové, že splňují rovnici  $H \times L = U$ . Vektor  $H$  podobně jako u rovinné želvy udává kam želva směřuje, tedy čelo. Vektor  $U$  směřuje směrem nahoru a vektor  $L$  pak 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: směr, podélný 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 ???

## 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 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 existujících větví. Tato vlastnost vychází ze základních výsledků při pozorování rostoucích rostlin. Nové rostlinné orgány, jako je stonek, větev, 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 pomoci jednoduchých D0L-systémů. Klasik

### 3.2.2 Mezotonický a Akrotonický vzor větvení

## 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 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 funkcionality, která pro přepisování vybere algoritmus, který je svými schopnostmi i časovou náročností nejvhodnější pro zpracováváný 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 a následně je vložit do L-systému kostry stromu. Podrobněji se touto částí funkcionality 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 interpret 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 řídicího modulu a je umístěn nad oběma 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 náležejí 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 zpracování 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 objektově 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 nim 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

### 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 rychlost a také na modularitu. Rychlost je zde důležitá zejména při iteračním procesu, neboť výsledné slovo může mít délku až v řádu milionů modulů. Je tedy třeba zajistit vhodné struktury pro ukládání dat a vybrat vhodný algoritmus pro rychlý přepis pravidel. Modularita a rozšiřitelnost zde s rychlostí úzce souvisí. Jednotlivé typy L-systémů 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, 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 interpretace. 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. Przemyslaw Prusinkiewicz 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 rozšiřitelnost a přehlednost.

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

- **Unikátní identifikátor L-systému** je nutné přiřadit každému L-systému. Ten se pak používá hlavně při vkládání subsystémů 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 APPENDIX
- **Subsystémy** lze vkládat 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 SUBSYSTEMY
- **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 výjimkou jsou stochastické systémy. Struktura pravidel je závislá na jejich typu a podrobněji je popsána v odpovídajících podkapitolách kapitoly TYPY.
- **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_Subsystems;
    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 `"/"`.

### 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 řídicími symboly XML a tak je zapotřebí nahradit je odpovídajícími XML entitami.

## 5.2 Soubory L-systémů

Při generování slova L-systémem dochází k vytváření velmi dlouhých řetězců s nestejnými 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 se zabývá kapitola [LINK](#)

### 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 často. 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 [LINK](#) ukazuje jakým je způsobem je struktura navržena.

Jedno políčko odpovídá jednomu bajtu. Každý modul obsahuje jeden znak, jenž jej identifikuje. Dále pak může obsahovat libovolný počet parametrů. Každý parametr je identifiková-

ván 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 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. Příklad takového 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 identifikátoru 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 funkcionality nalezení kontextu popsána v kapitole LINK.

### 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 rozšířitelnou funkcionality. Nabízí i možnosti optimalizace. Pro každý výraz je tedy vytvořena 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 LINK.

$$A(x, y) < B(z) > C(a, b, c) : (x + z \sim \text{abs}(a - c)) \rightarrow B(b - 1)FA(x/2, y/2) : \max(0, 1 - 1/z * z)$$

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.

Jelikož se následník skládá z klasických řetězců a výrazů, je jeho zpracování složitější. Jednoduché řetězce 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 paměťově nezabírá tolik prostoru jako `LongString`. 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 LINK

Z příkladu je vidět, že pokud po sobě následují dva výrazy, může být řetězec mezi nimi prázdný. Šedě jsou znázorněny znaky, které jsou zahozeny, neboť nejsou pro další zpracování potřebné. Při zpracování i následném přepisu pravidla se vždy začíná a končí. Velikost kontejneru s řetězci je tedy vždy o jednu větší než kontejner s výrazy. Je to z toho důvodu, že často existují pravidla, která obsahují v následníkovi pouze řetězec a žádný výraz.

## 5.4 Zpracování L-systémů

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

Abstraktní třída `AbstractLSystem` obsahuje několik metod, sloužících 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 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 vrací tato metoda jsou provedeny navíc finální úpravy. Jedná se o aplikaci pravidel homomorfismů a vložení slov, jež generují subsystémy tohoto L-systému.

Veškerá funkcionalita, jež je společná pro všechny typy L-systémů je implementována ve společném rodiči `LSystem`. Veškeré funkce se dají rozdělit do tří fází.

### 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 `bool 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 metoda `AbstractGenerator::createLSystem()`, 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.

#### 5.4.2 Inicializační fáze

Pravidla, která jsou předána jako řetězec, 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 `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 například pravidlo parametrického stochastického 0L-systému.

```
přidejPřepisovacíPravidlo( řetězec pravidlo )
{
}

```

#### 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 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 XX o typech větvených struktur, je nutné pro modelování pokročilejších botanických 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 prepisovacích pravidel. V následující kapitole jsou popsány 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.

$A \rightarrow A + (10) FBF$

Tento typ tedy nepodporuje zpracování výrazů. Každý následník se tedy 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 náhodnou 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 zapsaná již v podobě, jež 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 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.

$$A(x, y) < B(z) > C(a, b, c) : (x + z \sim \text{abs}(a - c)) \rightarrow B(b-1) FA(x/2, y/2) : \max(0, 1 - 1/z * z)$$

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 rekursivní volání sebe sama, aby všechny tyto situace byly korektně vyhodnoceny. Pokud dochází při ověřování k větvení, je většinou nutné najít 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 zásuvného modulu implementován také mechanismus pro zpracování dotazů. Podrobnějším popisem implementace dotazů se zabývá kapitola LINK.

## 5.6 Dotazy

Díky nim lze již během generování zjistit, kde a v jaké poloze se bude daný modul nacházet v prostoru. Díky rozšiřitelnosti 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 `QueryInterpreter`, což je speciálně uzpůsobený interpret pro zpracování dotazů. Jeho fungování je podobné jako u třídy pro interpretaci geometrie, jež bude popsána v kapitole [LINK](#). Narozdíl 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 dotazování 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 `QueryInterpreter` vyhodnocují. U dotazů, které si vynutí provedení interpretace slova je spolu s důvodem uvedena metoda `update()`. Ostatní dotazy obdrží odpověď okamžitě.

Instance třídy `QueryInterpreter` tedy při prvním volání dotazu `positionX()` zjistí všechny souřadnice 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 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 pravidla. 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 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 interpret 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 závorekám, věnuje se jim společná kapitola o implementaci zásobníku `LINK`.

## Kapitola 6

### Generování 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 interpretují a vytváří tak 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 již 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.

Pro interpretaci slov slouží abstraktní třída `AbstractInterpret`. Implementování jsou dva potomci. `GeometryInterpret` zpracovává všeskeré 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 [LINK](#).

Řetězec generovaný L-systémem je přetypován na nový typ. Z instance 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 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 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

### 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.

- ] — 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 dojít 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 interpretuje 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, vyrovnaní náklonu do vodorovné polohy nebo náhodné otočení v libovolném směru.
- 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 dané želvy na globální vnější vlivy. V případě gravitace se může například jednat o změnu pružnosti větve.

## Literatura

- [1] von Koch, H.: *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes*, Acta mathematica, 1905. 2.1
- [2] Lindenmayer, A.: *Mathematical models for cellular interaction in development, Díl I a II*, Journal of Theoretical Biology, 1968. 2, 2.3
- [3] Mitchinson, G. a Wilcox, M.: *Rules governing cell division in Anabaena*, Nature, 1972. 2.2
- [4] Prusinkiewicz, P. a Lindenmayer, A.: *The Algorithmic Beauty of Plants*, Springer-Verlag, 1990, 9780387972978.
- [5] Rozenberg, G. a Saloma, A.: *The mathematical theory of L-systems*, Academic Press, 1980. 2.2
- [6] Žára, J. a Beneš, B. a Sochor, J. a Felkel, P.: *Moderní počítačová grafika*, Computer Press, a.s., 2004.

**Příloha A**

**Příloha**