Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Vizualizace L-systémů pomocí želví grafiky

2022

Filip Kubíček

Vedoucí práce:
doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Filip Kubíček
Název práce:	Vizualizace L-systémů pomocí želví grafiky
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2022
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	49
Přílohy:	1 DVD
Jazyk práce:	český

Bibliograhic info

Author:	Filip Kubíček
Title:	L-systems visualisation using turtle graphics
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2022
Study field:	Applied Computer Science, full-time form
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	49
Supplements:	1 DVD
Thesis language:	Czech

Anotace

L-systémy jsou struktury podobné formálním gramatikám s řadou využití, zejména jako nástroje pro modelování rostlin. Pomocí želví grafiky je lze snadno vizualizovat. Předmětem této práce je desktopový program pro pohodlné definování a vykreslení i složitějších L-systémů.

Synopsis

L-systems are structures similar to formal grammars and have a range of applications, most notably modeling of plants. They are easily visualised by turtle graphics. This thesis presents a desktop program to conveniently define and visualise non-trivial L-systems.

Klíčová slova: L-systém; želví grafika; přepisovací pravidla; vizualizace

Keywords: L-system; turtle graphics; production rules; visualisation

Děkuji doc. RNDr. Miroslavu Kolaříkovi, Ph.D. za vedení této práce a za cenné rady.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Formální jazyky a s nimi související gramatiky patří k významným oblastem zájmu v oboru informatiky. Důležitým milníkem v jejich studiu bylo vytvoření Chomského hierarchie v 50. letech 20. století. Při práci s formálními gramatikami jsou klíčová tzv. přepisovací pravidla. Pomocí rekurzivních aplikací přepisovacích pravidel lze podle jednoduchých předpisů generovat i složitější výstupy.

S principy rekurze a přepisování se setkáme i v přírodě. V roce 1968 Aristid Lindenmayer vytvořil formalismus pro popis množení buněk jednoduchých organismů. Tyto Lindenmayerovy systémy, nebo též L-systémy, byly v následujících desetiletích dále studovány a rozšiřovány. Kromě využití L-systémů pro modelování jiných objektů než buněk – například celých rostlin – bylo též experimentováno s možnostmi vizualizace slov generovaných L-systémy. V tomto ohledu je důležitý rok 1986, kdy Przemyslaw Prusinkiewicz použil pro tento účel želví grafiku. [1]

Kombinace L-systémů s želví grafikou se ukázala být zajímavou, a tak v roce 1990 publikovali Prusinkiewicz a Lindenmayer knihu *The Algorithmic Beauty of Plants* [2], která se zabývá vytvářením grafických modelů rostlin pomocí L-systémů a želví grafiky. Tato kniha se stala dobrým výchozím bodem pro studium L-systémů a jejich vizualizace. Mnoho v ní uvedených poznatků se promítlo do způsobu, jakým je na L-systémy nahlíženo dnes.

I díky popularitě a důrazu na praktičnost výše uvedené knihy jsou dnes L-systémy relativně známým konceptem. Na internetu lze najít množství projektů, více či méně seriózních, jejichž smyslem je práce s L-systémy a vykreslení výsledků pomocí želví grafiky. Jedná se o jednoduché soukromé projekty, pluginy do již existujících programů nebo knihovny či aplikace pro vytváření komplexních modelů. Míra robustnosti těchto nástrojů často určuje jejich výhody a nevýhody. Zatímco jednodušší programy bývají snadno a intuitivně pochopitelné, často jsou schopné pracovat pouze s omezeným množstvím typů L-systémů a jejich výstupy nejsou příliš flexibilní. Na druhé straně robustní aplikace umožňují pokročilou práci s L-systémy, která může jít až za hranici formalismu L-systémů a zahrnovat prvky klasických programovacích jazyků. K dispozici mohou být také rozmanité možnosti práce s grafickým výstupem.[1]

Předmětem této práce je desktopový program pro práci s L-systémy a jejich vykreslování na základě želví grafiky, který byl pojmenován LSysVis. LSysVis by měl být schopný pracovat i se složitějšími typy L-systémů a poměrně přísně se držet jejich formalismu popsaném v knize *The Algorithmic Beauty of Plants*. Díky tomu by jeho silnou stránkou mělo být to, že umožní uživateli obeznámenému s teorií experimentovat s vlastními návrhy L-systémů bez omezení se na jednoduché příklady a bez nutnosti učit se používat nějakou složitější aplikaci. Možnosti grafického výstupu programu budou v porovnání se složitějšími aplikacemi střídmé, ale dostatečně flexibilní, aby bylo možné vykreslit i složitější typy

[1] Příkladem robustního projektu pro práci s L-systémy je například *L-Py*. Více informací o tomto projektu je k dispozici např. v jeho dokumentaci: https://lpy.readthedocs.io/en/latest/

L-systémů. Důraz je tedy kladen na to, aby bylo možné plnohodnotně pracovat s co nejširším množstvím L-systémů, a to ve dvourozměrném i trojrozměrném prostoru. Menší pozornost je věnována kosmetickým prvkům jako stínování, používání textur, integrování modelů např. listů nebo květů do výstupu programu apod.

2 L-systémy

L-systémy jsou struktury, které se podobají formálním gramatikám. Nejdůležitějším rozdílem je, že při jednom derivačním kroku jsou přepisovací pravidla použita paralelně na každý symbol slova.[2] Existují různé typy L-systémů, které lze zhruba zařadit do určité hierarchie podle jejich vyjadřovací síly. Vymezení určitého typu L-systému má často praktické důvody.

Cílem této kapitoly je představit typy L-systémů a s nimi spojené pojmy, se kterými program LSysVis pracuje. Hlavními zdroji pro tuto kapitolu byly *The Algorithmic Beauty of Plants* [2] a *L-systems: from the Theory to Visual Models of Plants* [3].

2.1 Deterministické bezkontextové L-systémy

Deterministické bezkontextové L-systémy, nebo také D0L-systémy, jsou základním typem L-systémů. D0L-systém je definován jako uspořádaná trojice $G = (\Sigma, \omega, \delta)$, kde Σ je neprázdná konečná množina symbolů, tzv. *abeceda*, $\omega \in \Sigma^+$ je neprázdné slovo nad abecedou, tzv. *axiom*, a $\delta \subset \Sigma \times \Sigma^*$ je konečná *množina přepisovacích pravidel*. Prvek množiny δ, tzv. *přepisovací pravidlo*, je tedy uspořádaná dvojice, jejímž prvním prvkem je symbol z abecedy Σ, tzv. *předchůdce*, a druhým prvkem je slovo nad abecedou Σ, tzv. *následovník*. Všimněme si, že následovníkem může být i prázdné slovo. U D0L-systémů platí, že pro každý symbol $a \in \Sigma$ existuje právě jedno přepisovací pravidlo, ve kterém je symbol a předchůdcem.

Přepisovací pravidla zapisujeme ve tvaru $a \to \Sigma^*$, kde na levé straně je předchůdce a na pravé straně je následovník. Pokud je v zápisu množiny přepisovacích pravidel vynecháno pravidlo s předchůdcem a, implicitně předpokládáme existenci pravidla $a \to a$.

Nechť $\alpha = a_1 \dots a_m$ je slovo nad abecedou Σ. Slovo $\beta = \pi_1 \dots \pi_m$ je *přímo generováno* slovem α, právě když pro každé $i = 1 \dots m$ platí $a_i \to \pi_i$. Tento vztah zapisujeme $\alpha \Rightarrow \beta$. Dále říkáme, že symbol a_i byl *přepsán* pomocí *aplikace* použitého přepisovacího pravidla. Slovo β tedy můžeme ze slova α vygenerovat tím, že přepíšeme všechny symboly slova α.

Při práci s L-systémem uvažujeme *aktuální slovo*. Změnu aktuálního slova nazýváme *derivačním krokem*. V jednom derivačním kroku je aktuální slovo α

[2] Toto je jen velmi obecné shrnutí principu L-systémů.

nahrazeno slovem β takovým, že $\alpha \Rightarrow \beta$. Na začátku, tedy před provedením prvního derivačního kroku, je aktuálním slovem axiom.

Uvažujme D0L-systém, který se skládá z axiomu:

$$\omega = a$$

a dvou přepisovacích pravidel:

$$a \to b$$
$$b \to aa$$

Aktuální slova při prvních derivačních krocích jsou:

$$0 : a$$
$$1 : b$$
$$2 : aa$$
$$3 : bb$$
$$4 : aaaa$$

2.2 Stochastické L-systémy

Stochastické L-systémy mohou mít více přepisovacích pravidel se stejným předchůdcem. Navíc je definována funkce $\rho : \delta \to \mathbb{N}^+$, která každému přepisovacímu pravidlu přiřadí kladné celé číslo, které určuje jeho *váhu*. Pravidlo s vahou w zapíšeme $a \to^w \Sigma^*$. Pokud není u pravidla uvedena jeho váha, předpokládáme, že má váhu 1. Při generování nového slova β ze slova α je pro každý symbol a slova α náhodně zvoleno přepisovací pravidlo s odpovídajícím předchůdcem. Pravděpodobnost, že pro přepsání symbolu a bude vybráno přepisovací pravidlo $P \in \delta$, je

$$\frac{\rho(P)}{\sum_{Q \in \delta, Q_1 = a} \rho(Q)},$$

kde Q_1 označuje předchůdce v pravidlu Q.

Jako příklad uvažujme L-systém s axiomem :

$$\omega = a$$

a přepisovacími pravidly:

$$a \to b$$
$$a \to^3 c$$

Po provedení prvního derivačního kroku bude v 25 % případů odvozeno slovo b a ve zbývajících 75 % případů bude odvozeno slovo c.

Výše uvedený L-systém je bezkontextový a stochastický. L-systémy této kategorie se označují 0L-systémy a je snadno vidět, že D0L-systémy jsou speciálním případem 0L-systémů.

2.3 Kontextově závislé L-systémy

Dalším rozšířením L-systémů je zavedení *kontextu* pravidel. Množina přepisovacích pravidel je u kontextově závislých L-systémů ve tvaru $\delta \subset \Sigma \times \Sigma^* \times \Sigma^* \times \Sigma^*$. Kromě předchůdce a následovníka mají přepisovací pravidla navíc *levý a pravý kontext*. Levý a pravý kontext jsou slova nad abecedou Σ. Kontextově závislá pravidla zapisujeme ve tvaru $\kappa < a > \lambda \to \Sigma^*$, kde κ je levý a λ pravý kontext pravidla. Pokud je některý z kontextů prázdné slovo, můžeme trochu nepřesně říct, že pravidlo tento kontext nemá, a ze zápisu slova ho vynechat.

Uvažujme přepisovací pravidlo P: $\kappa < a > \lambda \to b$ a slovo $\alpha = \mu a \nu$, kde $\mu, \nu \in \Sigma^*$. Při generování nového slova ze slova α je možné na symbol a aplikovat pravidlo P pouze při splnění podmínky, že κ je sufixem μ a λ je prefixem ν.

Kontextově závislé stochastické systémy označíme IL-systémy. Při stochastickém výběru přepisovacího pravidla bereme v potaz pouze ta pravidla, která splňují výše uvedenou podmínku kontextu. Také platí, že pravidla, která mají alespoň jeden kontext, mají vždy přednost před pravidly bez kontextu. Pokud tedy existují kontextová pravidla, která lze použít pro přepsání daného symbolu, je stochastický výběr uskutečněn pouze z těchto pravidel. Pokud není k dispozici žádné kontextové pravidlo, je výběr proveden z bezkontextových pravidel. V případě, že jsou definována pouze kontextová pravidla, ale žádné z nich nelze použít, použijeme implicitní pravidlo $a \to a$.

Uvažujme IL-systém s axiomem:

$$\omega = baaaaa$$

a přepisovacími pravidly:

$$\begin{aligned} b < a &\to ab \\ b &\to \\ aaaaa < b &\to x \end{aligned}$$

Dvě z pravidel jsou kontextová a pravidlo $b \to$ je bezkontextové. Prázdná pravá strana pravidla značí, že symbol b se přepíše na prázdné slovo. Z průběhu prvních několika derivačních kroků je vidět, že poslední kontextové pravidlo bylo aplikováno, jakmile byla splněna podmínka kontextu:

$$\begin{aligned} 0 &: baaaaa \\ 1 &: abaaaa \\ 2 &: aabaaa \\ 3 &: aaabaa \\ 4 &: aaaaba \\ 5 &: aaaaab \\ 6 &: aaaaax \end{aligned}$$

Upozorněme, že tento L-systém se chová zcela deterministicky, ačkoliv obsahuje dvě pravidla se stejným předchůdcem.

Součástí definice kontextově závislého L-systému může být i *množina ignorovaných symbolů* I. Symbolům v této množině se říká *ignorované symboly*. Pokud je v kontextu přepisovaného symbolu ve slově nějaký ignorovaný symbol, který neodpovídá kontextu aplikovaného přepisovacího pravidla, je možné tento symbol ve slově přeskočit. Uvažujme např. množinu ignorovaných symbolů $I = \{i\}$ a slovo xia. Pravidlo $x < a \rightarrow b$ lze v tomto případě aplikovat.

2.4 Parametrické L-systémy

U parametrických L-systémů uvažujeme, že slova, se kterými pracujeme, jsou posloupnostmi *parametrických symbolů* neboli *modulů*. Parametrický symbol má kromě svého jména i libovolný počet parametrů. Parametry jsou reálná čísla.[3] Pokud tedy máme abecedu jmen symbolů Σ, můžeme uvažovat množinu všech modulů $M = \Sigma \times \mathbb{R}^*$. Pokud má modul nenulový počet parametrů, zapisujeme je za jeho jméno do závorek a oddělujeme čárkami. Slovo $ab(1, 5.4)c$ se skládá ze tří modulů. Z toho prostřední modul b má dva parametry: 1 a 5.4.

Výše uvedenému typu parametrů budeme říkat *hodnotové parametry*. Při zápisu pravidel parametrických L-systémů uvažujeme navíc *formální parametry* a *pojmenovávací parametry*. Formální parametry jsou výrazy, které mohou obsahovat proměnné. Po dosazení do proměnných je možné formální parametry vyhodnotit a výsledkem je reálné číslo, tedy hodnotový parametr. Pojmenovávací parametry odpovídají jednotlivým proměnným. Levý kontext, pravý kontext a předchůdce pravidla jsou tvořeny moduly, které mohou mít pojmenovávací parametry. Moduly následovníka mohou mít formální parametry.

Princip aplikace parametrického vyhodnocovacího pravidla je vysvětlen na následujícím příkladu. Uvažujme pravidlo

$$k(x)l < a(y, z) \rightarrow b(x + y)c(x)d$$

a slovo

$$\alpha = fk(3)la(8, 9, 10)m(4, 5)n.$$

Modul $a(8, 9, 10)$ ve slově α se přepíše pomocí tohoto pravidla. Levý kontext pravidla – $k(x)l$ – odpovídá levému kontextu modulu $a(8, 9, 10)$ ve slově – $k(3)l$. V kontextu pravidla je navíc symbol $k(x)$ s pojmenovávacím parametrem x. Ve slově α tomuto pojmenovávacímu parametru odpovídá hodnotový parametr 3. Předchůdci pravidla, modulu $a(y, z)$, odpovídá ve slově modul $a(8, 9, 10)$. Dvěma pojmenovávacím parametrům y a z odpovídají první dva hodnotové parametry ve slově, tedy 8 a 9. Modul $a(8, 9, 10)$ má navíc ještě třetí hodnotový parametr 10, tomu však neodpovídá žádné pojmenovávací parametr, a proto z hlediska aplikace pravidla nemá význam. V opačné situaci, pokud by bylo pojmenovávacích parametrů více než hodnotových parametrů, nebylo by možné toto pravidlo

[3] Reálnými čísly je možné vyjádřit i logické hodnoty *False* (odpovídá číslu 0) a *True* (odpovídá číslu 1). Při zápisu modulu můžeme používat i toto značení.

aplikovat. Toto platí i pro moduly v levém a pravém kontextu: každému pojmenovávacímu parametru v pravidle musí odpovídat hodnotový parametr v původním slově, jinak nelze pravidlo aplikovat při generování nového slova.

Proměnným x, y, a z byly tedy přiřazeny hodnoty 3, 8 a 9. Následovník pravidla obsahuje moduly s formálními parametry. Do formálních parametrů tedy dosadíme získané hodnoty a dostaneme slovo s hodnotovými parametry. Toto slovo je výsledkem aplikace pravidla na přepisovaný modul. V uvedeném příkladu by se modul $a(8, 9, 10)$ přepsal na:

$$b(3 + 8)c(3)d = b(11)c(3)d$$

Ve formálních parametrech je možné použít základní matematické a logické operace a funkce. Možnosti programu LSysVis v tomto ohledu jsou popsány v kapitole 4.4.4.

Pravidlo parametrického L-systému může navíc obsahovat *podmínku*. Podmínka je, stejně jako formální parametr, výraz, který může obsahovat proměnné a je možné ho po dosazení vyhodnotit. Podmínku pravidla zapisujeme před šipku a uvozujeme ji dvojtečkou, např.:

$$a(x) : x > 10 \rightarrow b$$

Při aplikaci pravidla je po přiřazení hodnot proměnným podmínka vyhodnocena. Pokud je výsledkem hodnota s významem logické nepravdy, nelze toto pravidlo aplikovat.

Aby bylo pravidlo platné, musí být zaručeno, že při dosazování za proměnné ve formálních parametrech a v podmínce budou známy hodnoty všech proměnných. Hodnoty proměnných lze definovat i globálně a nemusí být nutné, aby byly všechny hodnoty proměnných získány spojením pojmenovávacích a hodnotových parametrů.

Podmíněná pravidla mají přednost před bezkontextovými i kontextovými pravidly. Mohou mít také váhu. Výběr pravidla pro přepsání daného modulu tedy probíhá nejprve z podmíněných pravidel. Pokud nelze žádné z nich aplikovat, probíhá výběr z kontextových pravidel. Pokud není k dispozici ani žádné aplikovatelné kontextové pravidlo, je výběr učiněn z bezkontextových pravidel. Pokud není explicitně definováno žádné bezkontextové pravidlo, použije se implicitní pravidlo přepsání modulu na sebe sama.

Uvažujme parametrický L-systém s axiomem:

$$\omega = a(3)$$

a přepisovacími pravidly:

$$a(x) \rightarrow a(x - 1)$$
$$a(x) : x == 0 \rightarrow c$$

Druhé pravidlo je podmíněné, a tak má přednost před prvním pravidlem. Dokud však není podmínka splněna, nemůže být podmíněné pravidlo použito. Prvních několik derivačních kroků vypadá takto:

$$\begin{aligned} &0 : a(3) \\ &1 : a(2) \\ &2 : a(1) \\ &3 : a(0) \\ &4 : c \end{aligned}$$

2.5 Stromová struktura v L-systémech

U všech doposud zmíněných L-systémů platilo, že slova jsou posloupnosti symbolů nebo modulů. Při vytváření modelů rostlin nemusí být tyto lineární struktury dostačující. Vhodnější je použít nějakou větvenou strukturu.

Základem pro tuto strukturu je *kořenový strom*, který se skládá z konečné množiny uzlů V a zobrazení, které každému uzlu $u \in V$ přiřadí množinu jeho *podstromů*, přičemž každý podstrom je podmnožinou V. Navíc platí, že strom je buď prázdný, nebo existuje právě jeden uzel, tzv. *kořen*, který neleží v podstromu žádného uzlu; libovolné dva podstromy jsou disjunktní a každý podstrom je opět stromem, pokud uvažujeme stejné zobrazení přiřazující uzlům jejich podstromy jako u celého stromu. Pokud je uzel v kořenem podstromu uzlu u, pak nazveme uzel v *potomkem* uzlu u a uzel u *rodičem* uzlu v.

Speciálním případem kořenového stromu je *osový strom*. V osovém stromu je nanejvýš jeden z podstromů uzlu u jeho *přímým podstromem*. Ostatní podstromy uzlu u jsou jeho *vedlejší podstromy*. Kořen přímého podstromu je *přímým potomkem* uzlu u a kořen vedlejšího podstromu je *vedlejším potomkem* uzlu u.

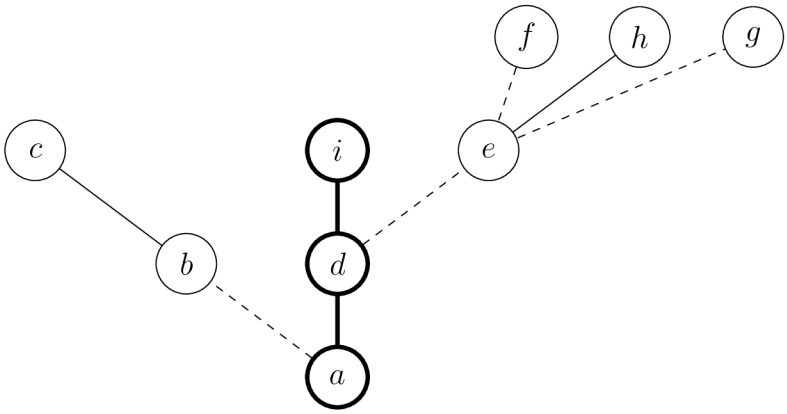
Posloupnost uzlů v osovém stromu nazveme *osou*, pokud:

- První uzel posloupnosti je kořenem celého stromu nebo je vedlejším potomkem nějakého uzlu.
- Každý následující uzel je přímým potomkem předchozího.
- Poslední uzel posloupnosti nemá žádné přímé potomky.

Ose, jejímž prvním uzlem je kořen celého stromu, říkáme *hlavní osa*.

Pokud uzly v osovém stromu ztotožníme se symboly nebo moduly L-systému, získáme vhodnou strukturu pro reprezentaci nelineárních slov. Při zápisu těchto slov zapíšeme nejdříve kořen stromu. Za každý právě zapsaný uzel u napíšeme nejprve jeho vedlejší podstromy. Každý vedlejší podstrom napíšeme do hranatých závorek. Poté zapíšeme přímý podstrom uzlu u, pokud existuje. Například struktura slova

$$a[bc]d[e[f][g]h]i$$



Obrázek 1: Slovo L-systému se stromovou strukturou.

je znázorněna na obrázku 1. Kořenový uzel odpovídá symbolu a. Přímí potomci jsou propojeni s rodiči plnou čarou a vedlejší potomci čarou čerchovanou. Hlavní osa stromu je zvýrazněna silnější čarou.

Generování nového slova probíhá ve stromových L-systémech podobně jako v těch lineárních. Následovníkem v pravidlu je slovo se stromovou strukturou. Uzel odpovídající předchůdci je nahrazen kořenem následovníka. Potomci uzlu předchůdce se stanou potomky posledního uzlu hlavní osy následovníka.

Pokud bychom na slovo znázorněné na obrázku 1 aplikovali pravidlo

$$e \to x[y]z,$$

výsledkem by bylo slovo

$$a[bc]d[x[y]z[f][g]h]i.$$

O něco komplikovanější je práce s kontextem. Levý kontext v pravidle musí být lineární posloupnost symbolů nebo modulů. Levým kontextem uzlu u ve slově je:

- Prázdna posloupnost, pokud u je kořenem celého slova.
- Jinak je to levý kontext rodiče uzlu u s rodičem uzlu u přidaným na konec.

Například levým kontextem uzlu h ve slově na obrázku 1 je posloupnost ade.

Pravý kontext v pravidle se může skládat z vedlejších podstromů a hlavního podstromu. Každému vedlejšímu podstromu v pravidle musí odpovídat nějaký vedlejší podstrom předchůdce ve slově. Hlavnímu podstromu musí odpovídat hlavní podstrom předchůdce ve slově. Osy podstromů ve slově přitom mohou být delší a uzly ve slově mohou mít vedlejší podstromy navíc.

Při zápisu pravého kontextu pravidla nejdříve uvedeme vedlejší podstromy a poté hlavní podstrom. Uveďme několik příkladů pravidel, která by byla apliko-

vatelná na symbol a slova na obrázku 1:

$$\begin{aligned} a > di &\rightarrow x \\ a > [b] &\rightarrow x \\ a > [bc] &\rightarrow x \\ a > [b]d[e] &\rightarrow x \\ a > [b]d[e]i &\rightarrow x \\ a > d[eh]i &\rightarrow x \\ a > d[e[f][g]] &\rightarrow x \end{aligned}$$

Při zápisu stromových slov je užitečné mít na paměti, že znaky [a] neoznačují moduly, ale mají speciální význam. Z praktických i technických důvodů se však vyplatí tyto znaky za symboly považovat. Z tohoto pohledu je i stromové slovo stále jen lineárním řetězcem modulů. Skutečného nelineárního významu slovo nabyde při aplikaci přepisovacích pravidel. Symboly [a] ovšem mají speciální pravidla: nelze je přepsat a nemohou mít parametry. Pokud by toto nebylo dodrženo, mohlo by při generování nového slova dojít k rozbití stromové struktury, kterou slovo reprezentuje.

3 Želví grafika

Želví grafika je způsob kreslení obrázků pomocí programem vykonávaných příkazů. Kreslení často probíhá na dvourozměrné plátno, ale želví grafiku lze rozšířit tak, aby bylo možné vytvářet modely v trojrozměrném prostoru. Modely jsou složeny z úseček, které navíc mohou mít další vlastnosti jako barvu a tloušťku. Modely jsou vytvářeny *želvou*. Želva je určena svou polohou (souřadnicemi v prostoru), orientací a dalšími vlastnostmi (např. právě zvolenou barvou). Tyto údaje o želvě souhrnně označujeme jako její *stav*. Pro účely kreslení L-systémů je v počátečním stavu želva umístěna do počátku soustavy souřadnic a její orientace je zvolena tak, aby želva směřovala směrem nahoru.

Želva následně přijímá příkazy a tím vytváří model. Základní typy příkazů jsou:

- Jdi dopředu ve směru své orientace o určitou délku a přitom kresli úsečku. Vlastnosti nově vytvořené úsečky jsou dány stavem želvy.
- Jdi dopředu ve směru své orientace o určitou délku bez vytvoření úsečky.
- Změň svou orientaci.
- Změň některou svou další vlastnost (např. aktuální barvu).
- Ulož svůj stav na zásobník.
- Načti svůj stav ze zásobníku.

Princip vizualizace L-systému pomocí želvy spočívá v tom, že slovo vygenerované L-systémem je procházeno od začátku po jednotlivých modulech a ty jsou interpretovány jako příkazy pro želvu. Klíčové přitom je, jaký význam jednotlivé moduly mají. Významy modulů v programu LSysVis jsou uvedeny v tabulce 3.

3.1 Vykreslovací pravidla L-systémů

Při interpretaci slova želvou platí, že jeden modul odpovídá jednomu příkazu želvy. To představuje omezení z hlediska možností vizualizace L-systému. Může být např. žádoucí, aby moduly s určitým jménem měly význam listů rostliny. List by přitom měl být vykreslen detailněji než jako úsečka. Z tohoto důvodu zavádíme v L-systémech *vykreslovací pravidla*. Tato pravidla se chovají stejně jako obyčejná pravidla, jediný rozdíl je v tom, kdy se aplikují. Předtím, než je slovo vygenerované L-systémem interpretováno želvou, je z tohoto slova vygenerováno pomocí vykreslovacích pravidel *vykreslovací slovo* pro želvu. Toto slovo se nestává aktuálním slovem L-systému a ani nijak jinak neovlivní další derivační kroky. Je využito pouze pro účely želvy. Vykreslovací pravidla zapisujeme se speciálním symbolem šipky. Pokud bychom např. chtěli, aby symbol l měl pro želvu význam několika příkazů najednou, mohli bychom zapsat následující vykreslovací pravidlo:

$$l \twoheadrightarrow [-F+F+F-|-F+F+F]$$

4 Uživatelská dokumentace

Tato kapitola popisuje, jakým způsobem se s programem LSysVis pracuje.

4.1 Spuštění programu

Program samotný lze spustit přímo, bez nutnosti instalace. Na počítači je však nutné mít nainstalovanou runtime knihovnu MSVC 2019 x64. Detailní instrukce pro zprovoznění programu, včetně seznamu požadavků na systém, jsou uvedeny v přiloženém souboru `readme.txt`.

4.2 Ukládání dat

Program ukládá data do registrů Windows. Jde o data týkající se uživatelského rozhraní jako velikost a pozice okna, umístění posledního otevřeného souboru atd. Pro smazání těchto dat je k dispozici skript `clearsettings.bat`. Po jeho spuštění dojde k vymazání dat z registrů a návratu do stavu před prvním spuštěním programu.

Soubory definic L-systémů jsou ukládány v textové formě s příponou `.lsys`. Jejich ukládání probíhá na základě příkazu od uživatele. Při ukládání souboru

s definicí L-systému je použito výchozí kódování dle nastavení operačního systému. Při otevírání souboru dojde k automatické detekci kódování.

4.3 Uživatelské rozhraní

Po spuštění programu je zobrazeno jeho hlavní okno. (Viz obrázek 2.) Nejdůležitějším prvkem hlavního okna je plátno, na které se vykresluje zadaný L-systém. Okno dále obsahuje lištu nabídek a stavový řádek. Ostatní prvky uživatelského rozhraní jsou řešeny formou doků, které lze umisťovat libovolně kolem plátna nebo i mimo hlavní okno. Je také možné měnit jejich velikost a jednotlivé doky skrývat.

Pro pohodlnou práci s programem je vhodné, aby si uživatel přizpůsobil rozmístění doků. Doky lze přemísťovat myší uchopením za horní lištu doku s jeho názvem a tažením. Velikost lze měnit po najetí myší na hranici doku podobným způsobem. Doky lze skrýt křížkem na jejich horní liště nebo v sekci *Window* lišty nabídek. Více v podkapitole 4.3.1.

Práce s programem probíhá typicky tak, že uživatel v doku *L-system definition editor* vytvoří nebo upraví definici L-systému a poté si ji pomocí doku *L-system simulation* nechá vykreslit na plátno.

Ve zbytku této podkapitoly jsou detailně popsány jednotlivé akce lišty nabídek a ostatní prvky uživatelského rozhraní.

4.3.1 Lišta nabídek

Lišta nabídek je rozdělena do několika sekcí, které se skládají z akcí. Níže je uveden popis jednotlivých sekcí.

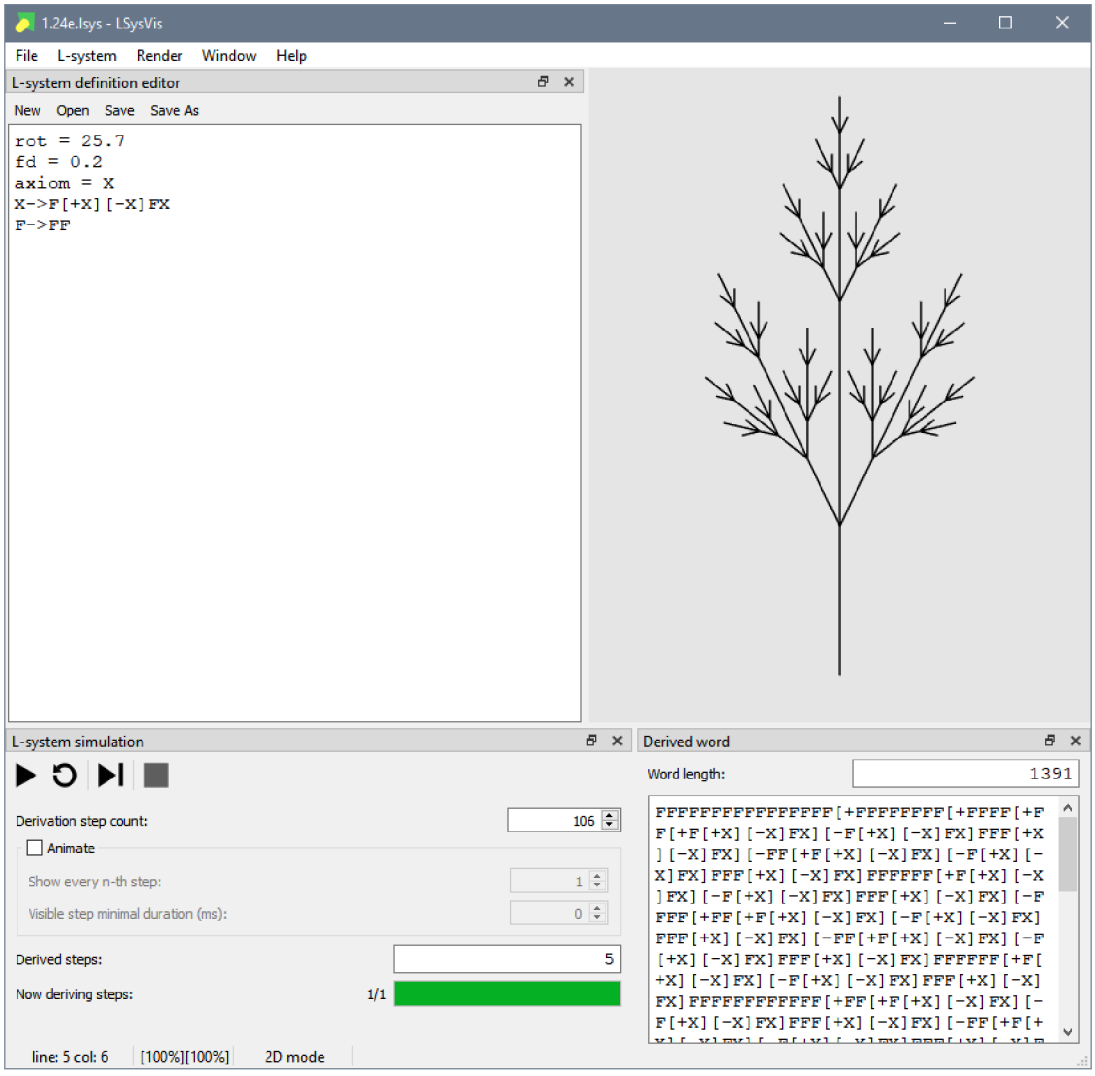
Sekce *File*

Akce *New*, *Open*, *Save* a *Save As* slouží k práci se souborem textové definice L-systému, která je načtená v doku *L-system definition editor*. Název právě otevřeného souboru je zobrazen v záhlaví okna programu. Po provedení změn v souboru se za jeho názvem zobrazí indikátor * a při pokusu o ukončení programu nebo otevření nového souboru se zobrazí výzva k uložení změn.

Vizualizaci L-systému vykreslenou na plátně lze exportovat. První možností je export ve formě rastrového obrázku formátu PNG či JPG — akce *Export image*. Po kliknutí na tuto akci se zobrazí okno, ve kterém je možné nastavit rozlišení obrázku. Je možné zvolit rozlišení s libovolným poměrem stran, výsledný obrázek je vždy správně škálovaný. Přepínačem *Show focus point* lze nastavit, zda v exportovaném obrázku má být vykreslen souřadnicový kříž. Toto nastavení je nezávislé na tom, zda je souřadnicový kříž vykreslen na plátně.

Druhou možností exportu je formát zdrojového kódu programu OpenSCAD[4]. Na rozdíl od předchozí možnosti exportu není výsledek ovlivněn aktuálním sta-

[4] https://openscad.org/



Obrázek 2: Uživatelské rozhraní programu LSysVis. Vpravo nahoře je plátno s vykresleným L-systémem. Dále jsou zobrazeny 3 doky, lišta nabídek nahoře a stavový řádek dole.

vem kamery, protože je exportován celý trojrozměrný model L-systému. V programu OpenSCAD pak lze s modelem dále pracovat. Například ho lze dále exportovat ve formátu STL vhodném pro 3D tisk.

Poslední akcí v části *File* je akce *Exit* — ukončení programu. Ukončení programu vyvolá též klávesová zkratka Alt+F4 nebo zavření hlavního okna programu.

Sekce *L-system*

Tato sekce obsahuje akce, které se týkají práce s L-systémem definovaným v doku *L-system definition editor*. Zejména jde o odvození dalšího derivačního kroku a vykreslení výsledku. Účinek akcí je ovlivněn údaji v doku *L-system simulation*. Více o tom, jak tyto akce fungují, je uvedeno v kapitole 4.3.5.

Sekce *Render*

V této sekci jsou akce, které souvisejí s vykreslením L-systému. Více o vykreslování L-systému je v kapitolách 4.3.2 a 4.6.

Akce *Reset camera* vrátí kameru do výchozí polohy.

Přepínač *Show focus point* určuje, zda je vykreslován osový kříž.

Přepínač *Automatically set camera mode* určuje, zda při načtení definice L-systému dojde k automatické změně režimu kamery. Vychází se přitom z toho, že L-systémy, které obsahují symboly změny úhlů tzv. pitch a roll, tedy symboly `^`, `&`, `\` nebo `/`, by měly být vykreslovány jako trojrozměrné. Při zaškrtnutí tohoto přepínače, a pokud L-systém obsahuje některý z těchto symbolů, bude automaticky zvolen režim kamery perspektivní 3D. V opačném případě bude kamera nastavena do režimu 2D. Poslední tři přepínače, *Camera 3D perspective*, *Camera 3D orthogonal* a *Camera 2D* slouží k manuální změně režimu kamery. Pokud je zaškrtnut předchozí přepínač *Automatically set camera mode*, je manuálně zvolený režim platný pouze do opětovného načtení definice L-systému. Režim ortogonálního 3D lze vybrat pouze manuálně.

Sekce *Window*

Tato sekce se sestává z několika přepínačů, přičemž každý přepínač odpovídá jednomu doku hlavního okna. Přepínače určují, zda je dok viditelný nebo skrytý. Doky lze skrýt i pomocí ikony křížku na jejich lištách. Skryté doky lze zobrazit pomocí těchto přepínačů.

Sekce *Help*

Součástí této sekce je akce *About*, která zobrazí okno s informacemi o programu a použitých knihovnách. Sekce dále obsahuje dva přepínače pro zobrazení nebo skrytí doků s nápovědou. Tyto přepínače jsou i v sekci *Window*.

4.3.2 Plátno

Plátno slouží k vykreslení L-systému. K vykreslení L-systému dojde po jeho definování[5] a odvození určitého počtu kroků[6].

Jakým způsobem je L-systém zobrazen, je dáno stavem tzv. kamery. Kameru si můžeme představit jako plochu, skrze kterou se díváme na vykreslený L-systém. Tato plocha je totožná s plátnem na obrazovce počítače a zároveň má své umístění v souřadnicovém systému vykresleného modelu. Ovládání kamery je navrženo tak, aby bylo jednoduché a intuitivní. Detailně je popsáno v následujících odstavcích.

Kamera programu LSysVis má tři režimy: perspektivní 3D, ortogonální 3D a 2D. Režim kamery lze kdykoliv manuálně přepnout, nebo povolit automatickou volbu režimu podle vykresleného L-systému[7].

Veškerý pohyb kamery probíhá kolem středového bodu, který je vždy umístěn do středu plátna. Kamera se tedy vždy „dívá“ na tento bod. Pokud je zaškrtnut přepínač *Show focus point* v sekci *Render* lišty nabídek, je středový bod vykreslen na plátně v podobě osového kříže. Kladná poloosa X je znázorněna červeně, Y modře a Z zeleně. Každá poloosa je reprezentována čárou délky 1. Tato velikost je vztažena k vykreslenému L-systému.

Pohyb kamery je umožněn interakcí myši s plátnem. Pomocí kolečka lze přiblížit nebo oddálit kameru od středového bodu. Pomocí uchopení pravým tlačítkem a tažením lze posunout středový bod. Posun probíhá po rovině rovnoběžné s rovinou obrazovky, která prochází středovým bodem. Ve 3D režimech jsou možnosti pohybu kamery rozšířeny. Uchopením levým tlačítkem a tažením lze rotovat kamerou kolem středového bodu. Ve 2D režimu je rotace uzamčena a levé tlačítko myši má stejnou funcki jako pravé – posouvání středového bodu. Ve 3D režimech lze navíc stisknutím kolečka a tažením posouvat středový pod po přímce procházející středem obrazovky a středovým bodem.

Kameru lze resetovat do výchozí polohy stiskem klávesy F4, volbou *Reset camera* v sekci *Render* lišty nabídek nebo pomocí tlačítka *Reset camera* v doku *Camera details*. Tento dok slouží také k získání informací o aktuálním stavu kamery.

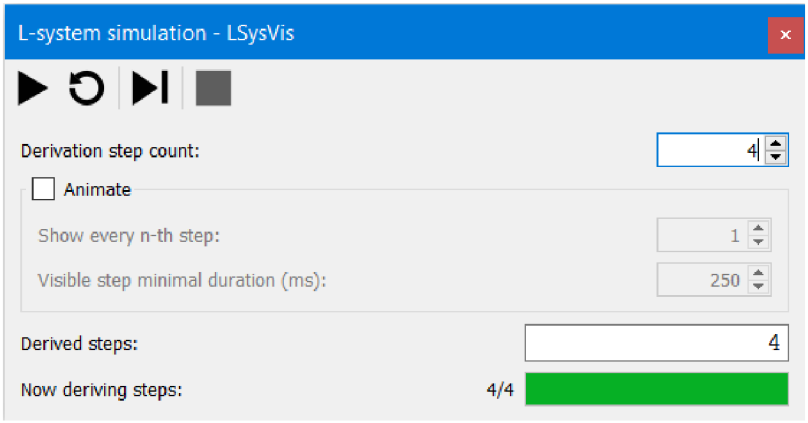
4.3.3 Dok *Camera details*

V tomto doku jsou zobrazeny informace o poloze kamery. Hodnota *Focus point* udává pozici středového bodu. *Rotation X* a *Rotation Y* uvádí rotaci kamery kolem středového bodu. *Focus point distance* je vzdálenost kamery od středového bodu a *Camera position* je pozice kamery. Přesněji jsou to souřadnice bodu, který odpovídá středu plátna.

[5] Více v kapitolách 4.3.4 a 4.4.

[6] Více v kapitole 4.3.5

[7] Více v posisu sekce *Render* lišty nabídek v kapitole 4.3.1



Obrázek 3: Dok *L-system simulation*.

4.3.4 Dok *L-system definition editor*

Tento dok obsahuje především textové pole, do kterého je zadávána definice L-systému. Specifikace jazyka pro definici L-systémů je v kapitole 4.4. Při práci s textem jsou k dispozici základní klávesové zkratky. Nabídku základních operací s přehledem příslušných klávesových zkratek lze zobrazit kliknutím pravým tlačítkem do textového pole. Informace o pozici kurzoru v textovém poli je k dispozici na stavovém řádku.

Kromě textového pole obsahuje dok toolbar s tlačítky pro práci se soubory definic L-systémů. Jejich funkce jsou totožné s akcemi v sekci *File* lišty nabídek.

4.3.5 Dok *L-system simulation*

Dok *L-system simulation* slouží k načtení definice L-systému zadané v doku *L-system definition editor*, odvození příslušného počtu derivačních kroků L-systému a následnému vykreslení výsledného slova na plátno.

Vzhled doku je vidět na obrázku 3.

V horní části doku je toolbar obsahující čtyři tlačítka reprezentovaná ikonami. Tlačítka odpovídají akcím v sekci *L-system* lišty nabídek. Jejich význam, v pořadí zleva doprava, je:

1. ▶ *Derive steps*

   Tato akce se skládá z několika kroků. Prvním krokem je načtení definice z doku *L-system definition editor*. V tomto kroku dochází k parsování definice a hlášení zjištěných chyb. V případě úspěšného parsování definice se aktuálním slovem učiní axiom načteného L-systému. Následuje provedení derivačních kroků. Jejich počet je dán hodnotou v poli *Derivation step count*. Po dokončení všech derivačních kroků je výsledné slovo zobrazeno na plátně.

   Pokud je zaškrtnut přepínač *Animate*, dochází navíc k vykreslení slov, která odpovídají jednotlivým derivačním krokům. Nemusí být vykresleny

všechny dílčí derivační kroky, je možné zobrazit jen každý n-tý derivační krok. K tomu slouží pole *Show every n-th step*. Pokud je uvedena hodnota 1, jsou postupně zobrazeny všechny derivační kroky. Slovo získané posledním derivačním krokem je nakonec zobrazeno vždy. Aby bylo možné kontrolovat rychlost animace, je k dispozici pole *Visible step minimal duration (ms)*. Hodnota v tomto poli udává minimální počet milisekund, po který bude každý vykreslený mezikrok viditelný. Pokud bude pro výpočet dalšího mezikroku, který se má zobrazit, potřeba více času než zadaný interval, může být animace pomalejší.

2.  *Reset simulation*

   Spuštění této akce provede parsování a, pokud nedojde k chybám, načtení definice L-systému. Pokud parsování skončí úspěšně, aktuálním slovem se stane axiom L-systému. Axiom je následně vykreslen na plátno.

3.  *Derive one step*

   Tato akce provede jeden derivační krok. Vychází přitom z aktuálního slova bez parsování a načtení definice L-systému, která mohla být od posledního načtení upravena. Nově odvozené slovo se stane aktuálním a je vykresleno na plátno.

4.  *Stop deriving*

   Tato akce zastaví právě probíhající derivaci. Pokud derivace právě neprobíhá, není akce dostupná. Touto akcí lze přerušit dlouho trvající výpočet jednoho derivačního kroku i dlouhou animaci mezikroků.

Ve spodní části doku se nacházejí dva informační prvky. Pole *Derived steps* informuje o tom, kolik derivačních kroků bylo potřeba k získání aktuálně odvozeného slova. Indikátor průběhu *Now deriving steps* ukazuje, kolik derivačních kroků bylo uživatelem zadáno k vykonání a kolik z nich již bylo provedeno. Každý derivační krok se může skládat z několika částí. O průběhu podčástí derivačních kroků informují ukazatele procent ve stavovém řádku[8].

Upozornění: při práci s některými L-systémy může počet modulů aktuálního slova velmi rychle narůstat. V takovém případě může dojít k zahlcení operační paměti počítače, což může v krajním případě vést až k pádu programu LSysVis.

4.3.6 Dok *Derived word*

Dok *Derived word* ukazuje aktuálně derivované slovo a jeho délku. Derivované slovo je zapsáno ve formátu jazyka programu LSysVis.

[8] Více v kapitole 4.3.9

4.3.7 Dok *Help: symbol interpretation*

Tento dok obsahuje přehled symbolů, které mají význam při interpretaci slova želví grafikou.

4.3.8 Dok *Help: special variables*

V tomto doku je přehled speciálních proměnných, které mají vliv na průběh derivace L-systému nebo ovlivňují jeho vykreslení.

4.3.9 Stavový řádek

Stavový řádek, umístěný vespod hlavního okna, se skládá ze tří informačních prvků.

První z nich informuje o pozici kurzoru v doku *L-system definition editor*.

Druhý ukazuje průběh aktuálního derivačního kroku. Jeden derivační krok může mít několik podčástí. Pokaždé je potřeba klasickým způsobem derivovat další slovo L-systému. Pokud má být daný krok vykreslen a pokud L-systém obsahuje vykreslovací pravidla, je potřeba provést ještě jednu derivaci pomocí těchto pravidel. Každé slovo, které se má vykreslit je navíc potřeba interpretovat pomocí želví grafiky. Z hlediska časové náročnosti jsou významné zejména první dvě podčásti. První indikátor procent v této části stavového řádku ukazuje průběh derivace dalšího slova L-systému a druhý indikátor ukazuje průběh derivace pomocí vykreslovacích pravidel.

Poslední prvek stavového řádku ukazuje aktuálně zvolený režim kamery.

4.4 Jazyk programu LSysVis

Program LSysVis pracuje s textovými definicemi L-systémů. Tyto definice jsou psány v jazyku, který byl navržen pro program LSysVis (dále jen „jazyk“) a vychází ze způsobu, jakým jsou L-systémy běžně zapisovány. [2][3] Cílem jazyka je být co nejpřirozenější a nejintuitivnější pro uživatele, kteří znají formální definici L-systémů.

Jeden textový soubor obsahuje vždy právě jednu definici L-systému. Definice L-systému se skládá z popisu přepisovacích pravidel, definování axiomu L-systému a případně dalších prvků ovlivňujících vlastnosti L-systému nebo jeho vizualizaci.

4.4.1 Struktura souboru definice

Jazyk používá tři základní syntaktické prvky. Jsou to zápisy derivačních pravidel, přiřazení hodnot globálním proměnným a komentáře. Příkladem definice L-systému s použitím všech základních prvků jazyka je zdrojový kód 1. Na řádku 6 je derivační pravidlo, které popisuje, jak se přepíše symbol `F`. Na řádcích 3 a 4 jsou přiřazení hodnot proměnným `rot` a `axiom`. Na řádcích 1, 3, 4 a 6 jsou komentáře.

```
1 # priklad jednoducheho L-systemu
2
3 rot = 30           # prirazeni hodnoty promenne
4 axiom = F          # prirazeni hodnoty promenne
5
6 F -> F[+F][-F]F    # prepisovaci pravidlo
```

Zdrojový kód 1: Příklad jednoduchého L-systému.

Komentáře jsou uvozeny znakem `#`. Veškerý text na řádku za tímto znakem (a včetně tohoto znaku) je považován za komentář.

Pomineme-li komentáře a bílé znaky, obecně platí, že každý řádek je buď prázdný, nebo odpovídá jednomu přiřazení proměnné, nebo jednomu přepisovacímu pravidlu.

V souboru definice jsou vždy nejdříve uvedena všechna přiřazení hodnot proměnným a poté následují přepisovací pravidla.

Proměnné `axiom` musí být vždy přiřazena hodnota. Definice však nemusí obsahovat žádná přepisovací pravidla. Nejmenší validní soubor definice L-systému má tedy jeden řádek s přiřazením do proměnné `axiom`.

4.4.2 Moduly a slova

Moduly a z nich složená slova tvoří části složitějších prvků jazyka. Při zápisu modulu je nejdříve zapsáno jeho jméno, a pokud má modul parametry, jsou uvedeny v závorkách za jménem a oddělené čárkami. Může mezi nimi být libovolný počet bílých znaků. Povolené znaky pro jméno modulu jsou:

- malá a velká písmena anglické abecedy
- číslice
- znaky `_[]+-^&\/|$.;!~‘`

Pokud je jméno symbolu delší než jeden znak, píše se celé do uvozovek. Např. `"dlouhe_jmeno"`.

Slovo je vytvořeno zápisem několika modulů za sebe. Mezi moduly mohou být bílé znaky. Uveďme příklad slova, které obsahuje parametrické i neparametrické moduly:

```
a"jmeno1"[b(5, 6)"dlouhe_jmeno"(9.5, -4)]
```

4.4.3 Názvy proměnných

S proměnnými se pracuje na úrovni pravidel v parametrech modulů, v podmínkách a s globálními proměnnými také na úrovni celého L-systému. Názvy proměnných se skládají z těchto znaků:

- malá a velká písmena anglické abecedy

- číslice
- znak _

První znak v názvu proměnné přitom nesmí být číslice.

4.4.4 Výrazy

Výrazy se používají k zápisu hodnotových a formálních parametrů, podmínek a při přiřazení do proměnných. Výrazy mohou obsahovat literály, proměnné, matematické nebo logické operace, funkce a závorky. Syntax a význam výrazů je stejný jako v běžných programovacích jazycích. Výrazy je možné závorkovat pomocí znaků `()` běžným způsobem.

Literály jsou celá a desetinná čísla a logické hodnoty `True` a `False`. Desetinná čísla se zapisují s desetinnou tečkou. Logické hodnoty lze použít i v rámci operací s čísly. Logická hodnota `True` se přitom považuje za číslo `1` a logická hodnota `False` za číslo `0`. Naopak při použití čísel jako logických hodnot se čísla, jejichž absolutní hodnota je větší rovna jedné, chovají jako logická pravda a ostatní čísla se chová jako logická nepravda.

Přehled podporovaných operátorů a funkcí:

- číselné unární operátory: `-`
- logické unární operátory: `!`
- číselné binární operátory: `+ - * / % ** << >>`
- logické binární operátory: `< > <= >= == != && ||`
- funkce:
  `sum(x,...)` `sqrt(x)` `sin(x)` `cos(x)` `tan(x)` `abs(x)` `pow(a, n)`

4.4.5 Přepisovací pravidla

Každé přepisovací pravidlo musí být zapsáno na samostatném řádku. Při zápisu přepisovacího pravidla je možné různé jeho části libovolně oddělovat bílými znaky.

Levá a pravá strana pravidla jsou odděleny šipkou `->`. Váha pravidla se zapisuje za šipku do závorek `->(3)`. Vykreslovací pravidla se zapisují speciální šipkou `->>`. I za touto šipkou může být v závorkách uvedena váha pravidla `->>(3)`.

V pravidle musí být uveden minimálně předchůdce a šipka:

```
A ->
```

Toto pravidlo přepíše symbol `A` na prázdný řetězec. Následník pravidla se píše na pravou stranu:

```
A -> BC"modul_XY"
```

Levý kontext pravidla se píše před předchůdce a je od něj oddělen znakem `<`. Pravý kontext se píše za předchůdce a je oddělen znakem `>`:

```
L < A~-> B
A > P -> B
L < A~> P -> B
```

Podmínka pravidla se píše před šipku a je uvozena dvojtečkou:

```
A(x): x<=0 -> B
A(x)> P: x<=0 -> B
```

Na závěr této podkapitoly uveďme příklad složitějšího přepisovacího pravidla:

```
H(x, foo)IJ < A(y) > K"el"(z): x+y>0 || foo -> X(1)Y(y*sin(z))Z
```

4.4.6 Přiřazení do globálních proměnných

Před přepisovacími pravidly jsou uvedeny přiřazení do globálních proměnných. Každé přiřazení do proměnné je zapsáno na jednom řádku. Nejprve je uvedeno jméno proměnné, poté znak = a nakonec hodnota proměnné. Mezi těmito prvky může být libovolný počet bílých znaků. Jakou hodnotu je možné přiřadit záleží na tom, o jakou proměnnou jde. Např. přiřazení čísla do proměnné `var_1` by vypadalo takto:

```
var_1 = 42.1
```

V této podkapitole byla popsána syntaktické stránka přiřazování do globálních proměnných. O tom, jak se s proměnnými v programu LSysVis pracuje, pojednává následující kapitola.

4.5 Proměnné

Přepisovací pravidla parametrických L-systémů mohou obsahovat proměnné, jejichž rozsah platnosti i životnost jsou omezeny vždy na jedno pravidlo, respektive jednu jeho aplikaci. Tyto proměnné považujeme za lokální proměnné. Definice L-systému obsahují také globální proměnné. Některé z nich mají speciální význam a mohou obsahovat jen speciální hodnoty. Číselné globální proměnné lze používat v přepisovacích pravidlech stejně jako ty lokální. Pokud pravidlo obsahuje pojmenovávací parametry, zastiňují při jeho aplikaci takto vzniklé lokální proměnné ty globální.

Všechny globální proměnné jsou konstanty. Jejich hodnota je vždy ta, která jim byla přiřazena na začátku. Při přiřazování hodnot číselným proměnným je možné používat výrazy, které obsahují dříve definované proměnné.

V následujících podkapitolách jsou popsány proměnné, které mají speciální význam. Většina speciálních proměnných má implicitní hodnoty. Seznam speciálních proměnných a jejich implicitních hodnot je v tabulce 1.

4.5.1 Speciální proměnná `axiom`

Hodnotou této proměnné je slovo s formálními parametry. Tyto formální parametry musí být možné vyhodnotit na hodnotové parametry. To znamená, že mohou obsahovat jen už dříve uvedené proměnné. Slovo, které vznikne po vyhodnocení

formálních parametrů, se při načtení definice L-systému stane jeho aktuálním slovem. Proměnné `axiom` musí být vždy přiřazena hodnota.

Příklad přiřazení do proměnné `axiom`:

```
axiom = AB(3)C(5)
```

4.5.2 Speciální proměnná `ignore`

Tato proměnná určuje množinu ignorovaných symbolů. Symboly, které mají být při kontrole kontextu ignorovány, se zapisují přímo za sebe a mohou být odděleny mezerami. Např.:

```
ignore = AB"modul_XY"C
```

4.5.3 Speciální proměnná `detail`

Jedná se o číselnou proměnnou. Její hodnota má význam z hlediska vykreslení L-systému na plátno. Úsečky modelu, který je vygenerován želví grafikou, se zobrazují jako pláště pravidelných hranolů. Hodnota detail udává, kolik stěn bude každý hranol mít. Pro tyto účely jsou desetinné hodnoty zaokrouhleny směrem dolů a hodnoty menší než 3 se interpretují jako hodnota 1, která způsobí, že místo pláště hranolu je zobrazen jen jeden obdélník.

4.5.4 Speciální proměnná `bg`

Tato proměnná určuje barvu pozadí plátna při vykreslování slov L-systému. Barvy se zadávají ve formátu RGB a rozsahy jednotlivých složek jsou 0 až 1. Větší a menší hodnoty jsou interpretovány jako jim nejbližší platné hodnoty. Jednotlivé složky se zapisují oddělené čárkami a libovolným počtem bílých znaků. Např.:

```
bg = 0.6, 0.7, 0.8
```

4.5.5 Speciální proměnné určující počáteční stav želvy

Proměnné `len`, `thick`, `rot` a `color` určují počáteční stav želvy při interpretaci slova. Princip, jak v programu LSysVis želva funguje, je vysvětlen v kapitole 4.6. Stav želvy se může při zpracování slova měnit, to však nemá na tyto proměnné vliv, protože jejich hodnoty jsou, stejně jako hodnoty všech globálních proměnných, konstantní. Proměnné `len`, `thick`, `rot` jsou číselné, hodnota `color` má tři složky a řídí se stejnými pravidly jako proměnná `bg`.

4.5.6 Speciální proměnné změn stavu želvy

Při interpretaci slov želvou lze měnit její stav. Některé příkazy přímo nastaví hodnotu požadované vlastnosti. Jiné příkazy přičtou k aktuální hodnotě vlastnosti předem danou inkrementační hodnotu[9]. Tuto inkrementační hodnotu určují

[9] Inkrementační hodnota se přičítá k hodnotě vlastnosti želvy, může ale být i záporná.

Tabulka 1: Speciální proměnné a jejich implicitní hodnoty.

Proměnná	Implicitní hodnota
`axiom`	
`ignore`	
`detail`	`25`
`bg`	`0.9, 0.9, 0.9`
`len`	`1.0`
`thick`	`0.2`
`rot`	`90.0`
`color`	`0.0, 0.0, 0.0`
`len_delta`	`-0.1`
`thick_delta`	`-0.1`
`rot_delta`	`10.0`
`color_delta`	`-0.1, -0.1, -0.1`

proměnné `len_delta`, `thick_delta`, `rot_delta` a `color_delta`. Z toho první tři jsou číselné a poslední má 3 složky stejně jako předchozí proměnné pracující s barvami. V posledním případě je přičtení provedeno po složkách.

4.5.7 Uživatelské proměnné

Kromě speciálních globálních proměnných lze definovat i vlastní globální proměnné. Uživatelské proměnné jsou vždy číselné[10]. Jejich časté využití je pro uložení konstant parametrických L-systémů, které se v jeho přepisovacích pravidlech objevují na více místech.

4.6 Interpretace slov pomocí želví grafiky

Po definování L-systému, odvození příslušného počtu derivačních kroků a případné aplikaci vykreslovacích pravidel je výsledné slovo interpretováno želví grafikou a zobrazeno na plátno.

Pro každé slovo je vytvořena nová želva, jejíž počáteční stav je dán globálními proměnnými z definice L-systému, případně jejich implicitními hodnotami, pokud nejsou v definici uvedeny. Počáteční stav želvy je popsán v tabulce 2.

Následuje zpracování výsledného slova. To je procházeno od začátku, modul po modulu. Každý modul odpovídá jednomu příkazu želvy. Význam některých modulů je ovlivněn i tím, kolik mají parametrů. V tom případě platí, že pokud má modul více parametrů než má pro želvu význam, jsou přebytečné parametry ignorovány. Zároveň je použita varianta příkazu, která bere v úvahu největší počet

[10]Mohou také mít hodnoty `True` a `False`. Ty jsou však zaměnitelné s číselnými.

Tabulka 2: Stav želvy v programu LSysVis.

Vlastnost	Počáteční hodnota
pozice	vždy v počátku soustavy souřadnic
orientace	vždy směrem nahoru, tedy ve směru kladné poloosy Y
délka kroku	hodnota proměnné `len`
úhel rotace při změně orientace	hodnota proměnné `rot`
tloušťka	hodnota proměnné `thick`
barva	hodnota proměnné `color`

uvedených parametrů. Moduly, které nemají pro želvu význam jsou ignorovány. Všechny moduly, které mají pro želvu význam, jsou uvedeny v tabulce 3.

4.7 Příklad L-systému v programu LSysVis

Zdrojový kód 2 je příkladem L-systému vytvořeného v programu LSysVis.

```
1  bg = 1, 1, 1
2  rot = 35
3  color = 0.4, 0.2, 0.1
4
5  axiom = X
6
7  X -> F[+X]F[-X]+(20)X
8  F -> FF
9
10 X ->> ;(0,0.7,0)F
```

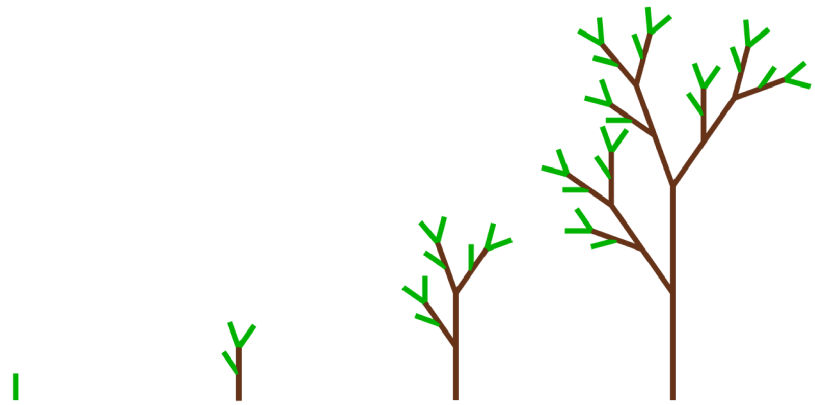
Zdrojový kód 2: Příklad L-systému definovaného v programu LSysVis.

Přiřazení do proměnné `bg` na řádku 1 nastaví barvu pozadí na bílou. Přiřazení na řádcích 2 a 3 určují počáteční stav želvy. Nastaví její implicitní úhel otočení a barvu. Na řádku 5 je definován axiom L-systému, v tomto případě je to slovo délky jedna. Na řádcích 7 a 8 jsou přepisovací pravidla, která způsobí postupný růst L-systému. U pravidla na řádku 7 je zajímavé, že jeden ze symbolů, které způsobují rotaci želvy, má parametr `20`. Při jeho interpretaci se želva otočí o 20 stupňů, zatímco ostatní symboly `+` a `-` způsobí rotaci želvy o `35`, jak je dáno stavem želvy. Na řádku 10 je vykreslovací pravidlo, které dodá symbolu `X` grafickou podobu. Před zpracováním želvou jsou symboly `X` ve slově nahrazeny následníkem `;(0,0.7,0)F`, jehož modul `;(0,0.7,0)` nastaví barvu na zelenou a následně symbol `F` nakreslí čáru.

Na obrázku 4 je axiom a první 3 derivační kroky takto definovaného L-systému, vykreslené programem LSysVis.

Tabulka 3: Význam modulů při interpretaci želvou.

Modul	Význam pro želvu
`F`	Jdi dopředu o délku kroku danou stavem želvy. Přitom vytvoř čáru, jejíž tloušťka a barva jsou dány stavem želvy.
`F(x)`	Jdi dopředu o délku kroku danou parametrem `x`. Přitom vytvoř čáru, jejíž tloušťka a barva jsou dány stavem želvy.
`f`	Jdi dopředu o délku kroku danou stavem želvy.
`f(x)`	Jdi dopředu o délku kroku danou parametrem `x`.
`[`	Ulož stav želvy na zásobník.
`]`	Načti stav želvy ze zásobníku.
`+`	Změň orientaci rotací úhlu *yaw* v kladném směru.
`-`	Změň orientaci rotací úhlu *yaw* v záporném směru.
`^`	Změň orientaci rotací úhlu *pitch* v kladném směru.
`&`	Změň orientaci rotací úhlu *pitch* v záporném směru.
`\`	Změň orientaci rotací úhlu *roll* v kladném směru.
`/`	Změň orientaci rotací úhlu *roll* v záporném směru.
U výše uvedených modulů je úhel rotace dán stavem želvy. Pokud mají tyto moduly alespoň jeden parametr, udává tento parametr úhel rotace. Úhly se zadávají ve stupních.	
`|`	Otoč se. Ekvivalentní s `+(180)` a `-(180)`.
`$`	Prováděj rotaci změnou úhlu *roll*, dokud není želva horizontálně vyrovnána. Detaily toho, jak tento příkaz funguje jsou popsány v [2].
`` ` ``	Přičti k aktuální délce kroku hodnotu proměnné `len_delta`.
`!`	Přičti k aktuální tloušťce hodnotu proměnné `thick_delta`.
`~`	Přičti k aktuálnímu úhlu rotace hodnotu proměnné `rot_delta`.
`;`	Přičti k jednotlivým složkám aktuální barvy hodnoty proměnné `color_delta`.
`` `(x) ``	Nastav aktuální délku kroku na hodnotu parametru `x`.
`!(x)`	Nastav aktuální tloušťku na hodnotu parametru `x`.
`~(x)`	Nastav aktuální úhel rotace na hodnotu parametru `x`.
`;(r, g, b)`	Nastav aktuální barvu podle hodnot parametrů `r, g, b`.



Obrázek 4: Příklad L-systému vykresleného programem LSysVis.

V adresáři `data/` jsou ukázkové soubory definic dalších L-systémů. Některé z nich jsou s drobnými úpravami převzaté z literatury, jiné byly vytvořeny při experimentování v programu LSysVis. Zdroje jsou vždy uvedeny ve formě komentářů v daném souboru.

5 Programátorská dokumentace

Program LSysVis byl vytvořen ve vývojovém prostředí *Qt Creator* v jazyce C++ a za použití frameworku *Qt*[11]. Program byl vyvíjen a testován na operačním systému *Windows 10*. Další knihovny a nástroje, které program využívá, jsou: *OpenGL Mathematics (GLM)*[12], *Cparse*[13] a *ANTLR v4*[14]. K renderování želví grafiky je využito API *OpenGL* verze 3.3 *Core Profile*.

5.1 Struktura projektu

Adresář `src/LSysVis/` na přiloženém DVD odpovídá kořenovému adresáři projektu. V tomto adresáři je důležitý soubor `LSysVis.pro`, který obsahuje konfiguraci projektu.

Program lze zhruba rozdělit do čtyř částí. Hlavní z nich je část tvořící uživatelské rozhraní a spojující funkcionalitu ostatních částí. Zdrojové soubory této části programu jsou přímo v podadresáři `src/` kořenového adresáře projektu. Část starající se o parsování definic L-systémů je v adresáři `src/parse/`. V adresáři `src/lsystem/` jsou soubory, které tvoří celek umožňující práci s L-systémy, a v adresáři `src/render/` je část programu zajišťující želví grafiku a grafický

[11] https://www.qt.io/
[12] https://glm.g-truc.net/
[13] https://github.com/cparse/cparse
[14] https://www.antlr.org/

výstup. Tyto části nejsou striktně odděleny. Celky pro parsování a renderování L-systému pracují s objekty definovanými v `src/lsystem/`.

V kořenovém adresáři projektu jsou dále podadresáře `lib/`, ve kterém jsou zdrojové soubory knihoven Cparse a GLM, a `res/`, který obsahuje soubory ikon a shaderů, se kterými program pracuje.

5.2 Kompilace

K sestavení projektu je využíván systém *qmake*, který je integrován v prostředí Qt Creator. Používaným překladačem je MSVC2019 64bit. K překladu projektu byl původně používán překladač MinGW 64bit, ale kvůli problémům[15] s fungováním tohoto překladače pod Windows 10 došlo k jeho změně.

Při překladu programu pomocí vývojového prostředí se o nalezení hlavičkových souborů knihoven a o linkování postará systém qmake díky konfiguraci v souboru `LSysVis.pro`. Nástroj ANTLR je však potřeba řešit externě. V adresáři `src/antlr4-cpp-runtime-4.9.3-source/` na přiloženém DVD je projekt, který obsahuje zdrojové kódy pro runtime ANTLR. Tento projekt je nutné zkompilovat a propojit s projektem LSysVis. Pro kompilaci je nejjednodušší projekt ANTLR otevřít ve Visual Studiu 2019 a zkompilovat. K propojení takto vzniklých objektových souborů a hlavičkových souborů s projektem LSysVis slouží direktivy `LIBS`, `INCLUDEPATH` a `DEPENDPATH` na posledních řádcích konfiguračního souboru `LSysVis.pro`.

Kromě propojení runtime části nástroje ANTLR je nutné do projektu přidat zdrojové soubory jazyka C++, které obsahují třídy pro práci s gramatikou popsanou nástrojem ANTLR. Tyto třídy jsou generovány projektem umístěným v adresáři `src/antlr4 Grammar/`. Jedná se o projekt prostředí Visual Studio Code. Projekt vyžaduje instalaci doplňku pro práci s nástrojem ANTLR. Zdrojové soubory vygenerované do `src/antlr4 Grammar/output` je nutné překopírovat do podadresáře `src/parse/antlr4` projektu LSysVis.

Po zajištění těchto závislostí je možné projekt LSysVis v prostředí Qt Creator zkompilovat. Poté je vhodné vzniklý soubor `LSysVis.exe` přesunout do nové složky a pomocí nástroje *windeployqt*, který je součástí instalace prostředí Qt Creator, přidat do této složky požadované soubory dynamicky načítaných knihoven. Ručně je pak potřeba přidat soubor `antlr4-runtime.dll` vytvořený překladem runtime projektu ANTLR. Tímto procesem vznikne samostatně spustitelný program LSysVis tak, jak je k nalezení v adresáři `bin/` na přiloženém DVD.

5.3 Hlavní část programu a uživatelské rozhraní

V této kapitole je popsán význam souborů umístěných přímo v adresáři `src/`. Jsou to soubory tříd, které zajišťují uživatelské rozhraní a pracují s ostatními

[15] Např. chyba související s generováním náhodných čísel pomocí `std::random_device`. O této chybě se zmiňuje i dokumentace jazyka C++: https://en.cppreference.com/w/cpp/numeric/random/random_device

částmi programu na základě vstupu od uživatele.

Uživatelské rozhraní programu je postaveno na frameworku Qt. To do velké míry určuje jeho strukturu. Jednotlivé prvky uživatelského rozhraní jsou většinou realizovány děděním některé třídy definované knihovnou. Pro komunikaci mezi jednotlivými prvky je využíván systém *signálů* a *slotů* typický pro Qt. Logika programu je oddělena od vzhledu.

Pro vytvoření vzhledu aplikace byl použit nástroj *Qt Designer*, který je součástí vývojového prostředí. K definování např. widgetu, který je zodpovědný za obsah některého doku, je vytvořena třída, která dědí z třídy `QWidget`. S ní je propojen soubor s příponou `.ui`, ve kterém je popsán vzhled widgetu. Obsah souboru je vytvořen pomocí grafického editoru Qt Designer.

Ve třídě, která takto vznikla, je implementována logika interakce grafického prvku s ostatními částmi programu. Samotná data, se kterými třída pracuje, jsou většinou uložena v dalších objektech. Můžeme tedy rozlišit *view* – popis grafické stránky prvku, *controller* – třídu dědící z některé třídy knihovny a implementující interakci se zbytkem programu a *model* – třídu starající se o uložení dat.

5.3.1 Třída `LSystemModel`

Objekt této třídy je vytvořen na začátku běhu programu a stará se uložení aktuálního stavu L-systému načteného programem. Jak napovídá jeho název, tento objekt odpovídá modelu ve výše uvedené hierarchii. S tímto objektem pracuje celá řada prvků uživatelského rozhraní. K tomu je používán systém signálů a slotů. Vlastníkem tohoto objektu je po celou dobu jeho života objekt hlavního okna. To odpovídá tomu, že objekt existuje po celou dobu běhu programu a jednotlivé třídy typu controller s ním komunikují, aniž by se dalo říci, že některé z nich tento objekt patří více než ostatním.

Objekt `LSystemModel` vlastní objekty `LSystemSimulationWorker` a `LSystemRenderWorker`. Tyto objekty jsou po vytvoření v konstruktoru objektu `LSystemModel` přesunuty do vlastních vláken – objektů třídy `QThread`. Poté je jim možné posílat signály, jejichž obsluhy (metody těchto `Worker` objektů) se vykonávají v těchto vláknech. `LSystemModel` tímto způsobem může např. přijmout signál, že má dojít k provedení 10 derivačních kroků. Tento signál je dále poslán objektu `LSystemSimulationWorker`, který ve svém vlákně spustí práci. `LSystemSimulationWorker` poté vysílá signály o odvození jednotlivých slov zpět objektu `LSystemModel`, který je vysílá dále. Ostatní prvky uživatelského rozhraní mohou naslouchat signálům od `LSystemModel`, pokud o ně mají zájem.

`LSystemSimulationWorker` zajišťuje práci s L-systémem, tedy práci s objekty definovanými v `src/lsystem/`. `LSystemRenderWorker` se stará o zpracování odvozeného slova želví grafikou. Tyto třídy však neimplementují tyto funkce, pouze se starají o jejich vykonání voláním metod jiných objektů.

5.3.2 Běh programu

Při spuštění programu se zavolá funkce `main`, která je definovaná v souboru `main.cpp`. Pro prologu programu, při kterém je zejména vytvořena instance třídy `QApplication`, dojde k vytvoření hlavního okna programu, tedy objektu třídy `MainWindow`. V konstruktoru `MainWindow` dojde k vytvoření všech ostatních prvků uživatelského rozhraní a objektu typu `LSystemModel`. Následně jsou propojeny signály a sloty jednotlivých objektů. Po vykonání konstruktoru hlavního okna je běh programu předán smyčce událostí, která běží, dokud nedojde k ukončení programu.

Mezi objekty, které jsou vytvořeny v konstruktoru hlavního okna, patří kromě objektů prvků uživatelského rozhraní i objekty třídy `QAction`. Jsou to objekty, které odpovídají uživatelským akcím, jako je otevření souboru nebo načtení definice L-systému. Objekt této třídy sjednocuje spuštění dané akce z různých míst: např. stiskem tlačítka nějakého widgetu, pomocí lišty nabídek nebo klávesovou zkratkou. Signály akcí jsou pak napojeny na sloty objektů, které jsou zodpovědné za danou činnost. Např. signál akce otevření souboru je napojen na controller widgetu *L-system definition editor* a signál akce pro změnu režimu kamery je napojen na slot widgetu plátna.

5.4 Simulace L-systému

Tato kapitola se věnuje třídám implementujícím datové struktury a logiku pro práci s L-systémy, které jsou jsou ve složce `src/lsystem/`.

5.4.1 Třídy pro práci s L-systémy

Základní třídou pro uložení dat L-systému je třída `Symbol`. Jedná se o generickou třídou, jejíž typ je upřesněn tím, jaký typ parametrů symbol má. Pracujeme se třemi konkrétními typy parametrů, jejichž definice jsou ve zdrojovém kódu 3.

```
1 using SymbolBind = Symbol<std::string>;
2 using SymbolExpr = Symbol<cparse::calculator>;
3 using SymbolActual = Symbol<cparse::packToken>;
```

Zdrojový kód 3: Typy symbolů v programu LSysVis.

Na řádku 1 je definován typ symbolu s pojmenovávacími parametry, kde každý parametr odpovídá textovému řetězci – názvu proměnné. Na řádku 2 je definice typu symbolu s formálními parametry. Objekt typu `cparse::calculator` představuje zkompilovaný výraz, který může obsahovat proměnné a je možné ho vyhodnotit. Jedná se objekt knihovny Cparse, která je v projektu použita pro práci s výrazy, které je potřeba dynamicky vyhodnocovat. Jak je vidět na řádku 3, symboly s hodnotovými parametry také využívají objekty knihovny Cparse. Objekt `cparse::packToken` představuje jednu hodnotu, kterou může být celé nebo

desetinné číslo a logická hodnota[16]. Hodnoty v knihovně Cparse jsou dynamicky typované, což může být pro uživatele příjemnější, ale zároveň lze z uživetelského hlediska hodnoty stále považovat za reálná čísla. Více o tom, jak jsou různé typy hodnot interpretovány jako reálná čísla, je v kapitole 4.4.4.

Třída `Symbol` obsahuje datové členy pro uložení parametrů a také číselného ID symbolu. To je dané typem `SymbolID`. Převod jmen symbolů na jejich ID je proveden při parsování a interně jsou symboly rozlišovány pomocí jejich čísel.

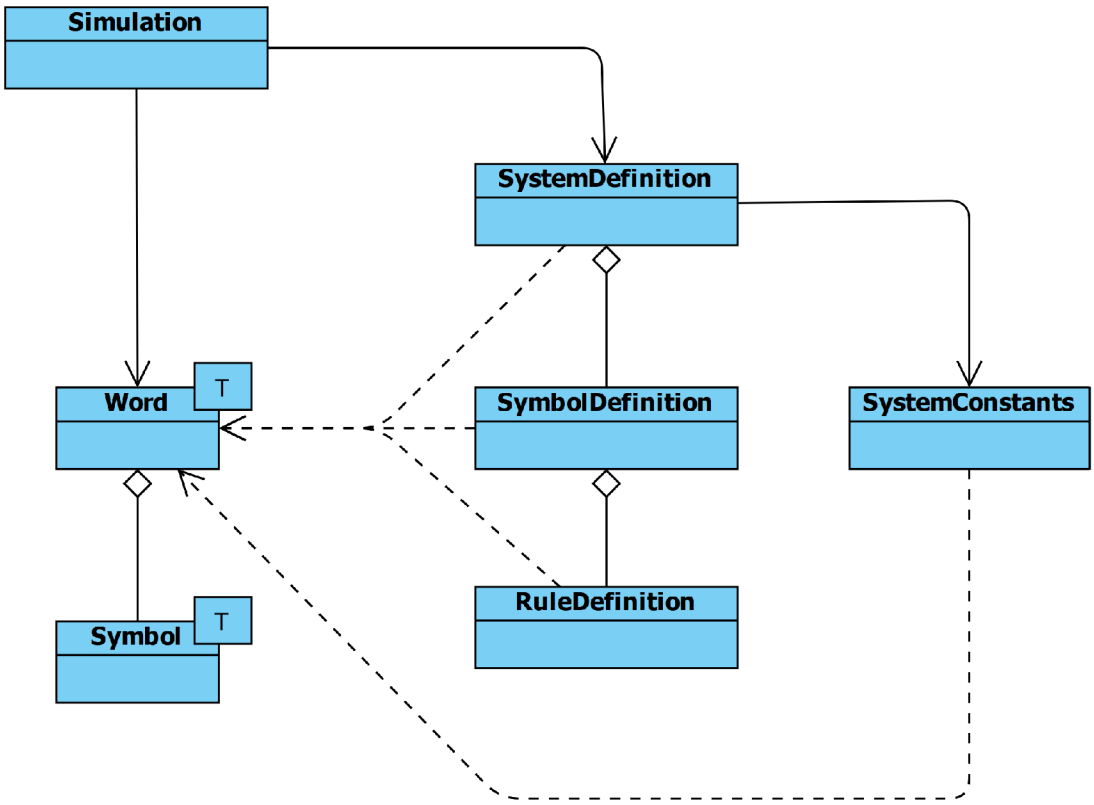
Ze symbolů se skládají slova, která jsou reprezentována třídou `Word`. Jedná se opět o generickou třídu, která nabývá třech konkrétních typů podle toho, z jakých symbolů se slovo skládá. Ve třídě `Word` jsou symboly uloženy jednoduše pomocí objektu `std::vector`. Třída `Word` navíc umožňuje práci se slovem jako se stromem. Stromová struktura je daná symboly `[` a `]`. Objekt typu `Word` si udržuje indexy odpovídajících si závorek a tím umožňuje efektivní práci se stromovou strukturou slova. S tím souvisí i kontrola vyváženosti závorek, která je prováděna při práci s třídou `Word`.

Třídy `Symbol` a `Word` jsou základními datovými strukturami pro práci s L-systémy, které jsou často využívány ostatními třídami. Jednou ze tříd, která tyto struktury využívá, je třída pro uložení přepisovacího pravidla. Jedno přepisovací pravidlo odpovídá jednomu objektu typu `RuleDefinition`. Tento objekt obsahuje veškeré informace potřebné pro aplikování pravidla na odpovídající symbol, nebo rozhodnutí, že k aplikaci nemůže dojít kvůli nesplnění podmínky kontextu nebo podmínky parametrického pravidla. Objekt třídy `SymbolDefinition` pak obsahuje pravidla jednoho symbolu (všechna pravidla, jejichž předchůdce je modul s daným jménem). Pro každý symbol jsou vytvořeny dva objekty typu `SymbolDefinition`: jeden pro obyčejná pravidla a druhý pro vykreslovací pravidla. Definice pravidel roztříděné podle svých předchůdců jsou uloženy ve třídě `SystemDefinition`. V té jsou kromě pravidel L-systému uloženy i další parametry definice L-systému jako axiom nebo popis počátečního stavu želvy při interpretaci. O ukládání těchto hodnot se stará třída `SystemConstants`. Třída `SystemDefinition` tedy obsahuje celou definici L-systému, jak je zadána uživatelem. Práci s takto definovaným L-systémem zajišťuje třída `Simulation`, která obsahuje především objekt definice L-systému a jeho aktuální slovo. V této podkapitole byly uvedeny hlavní třídy pro práci s L-systémy a jejich vzájemné vztahy, které jsou zachyceny i na obrázku 5.

5.4.2 Provedení derivačního kroku

V této podkapitole předpokládáme, že existuje objekt třídy `Simulation` obsahující definici L-systému, tedy objekt `SystemDefition`, a nějaké aktuální slovo, ať už jde o axiom převzatý ze `SystemDefition` nebo dříve odvozené slovo. Jak dojde ke vzniku takového objektu `Simulation` je popsáno v kapitole 5.5.

[16]Cparse podporuje více datových typů. Pro účely programu LSysVis však není žádoucí, aby byla práce s ostatními datovými typy umožněna. Proto je typ hodnot při běhu programu kontrolován a uživatel je upozorněn, pokud není typ hodnoty povolený. Ke kontrolám dochází zejména po vyhodnocení výrazů během aplikace přepisovacích pravidel nebo parsování.



Obrázek 5: Diagram tříd pro práci s L-systémy.

K provedení derivačního kroku dojde po zavolání metody `deriveNextWord` třídy `Simulation`. Podobný efekt má i volání metody `deriveRenderWord`, obě totiž používají privátní metodu `doDeriveWord`.

Před aplikací přepisovacích pravidel na aktuální slovo je zavolána metoda `setStepCommonValues` třídy `SystemDefition`. Ta způsobí zavolání stejnojmenné metody objektů `SymbolDefinition` a ty opět zavolají stejnojmenné metody objektů `RuleDefinition`. Tímto způsobem se k objektům `RuleDefinition` dostanou informace o derivačním kroku, který se bude provádět. Zejména to je aktuální slovo, ze kterého bude generováno nové slovo, a proměnné L-systému.

Následně je v metodě `doDeriveWord` aktuální slovo procházeno symbol po symbolu. Z objektu `SystemDefition` je vždy vybrán objekt `SymbolDefinition` odpovídající danému symbolu. Následně je zavolána metoda `deriveSuccessor` tohoto objektu, ve které dojde k nalezení aplikovatelného pravidla daného symbolu a k jeho aplikaci. Rozhodnutí, zda je pravidlo na daný symbol ve slově aplikovatelné, a případnou aplikaci pravidla zajistí metoda `apply` objektů typu `RuleDefinition`. Aplikace pravidla se provede přidáním následníka pravidla na konec slova, které bylo nově vytvořeno na začátku metody `doDeriveWord`.

Nakonec, po derivování nového slova, se toto slovo stane aktuálním slovem, nebo je vráceno jako vykreslovací slovo.

5.4.3 Aplikace přepisovacího pravidla

Při volání metody `apply` třídy `RuleDefinition` dojde k pokusu o aplikaci přepisovacího pravidla. Zda byl pokus úspěšný ukazuje návratová hodnota.

Při vytvoření objektu `RuleDefinition` jsou v něm uloženy pravdivostní hodnoty `requiresContextCheck` a `requiresNamespaceBinding`. Ty jsou při aplikaci pravidla použity k rychlému rozhodnutí, co všechno je potřeba vykonat. Pokud jsou obě nastaveny na `false`, dojde jen k překopírování následovníka pravidla do nově vznikajícího slova.

Jinak se pokračuje ke kontrole levého a pravého kontextu, pokud to pravidlo vyžaduje. Algoritmus pro kontrolu pravé části kontextu je o něco rozsáhlejší kvůli složitějšímu způsobu práce se stromovou strukturou slova. Při kontrole kontextů vzniknou také dva seznamy ukazatelů na moduly v původním slově. Pořadí odkazů v těchto seznamech odpovídá pořadí modulů v levém a pravém kontextu pravidla. Každý ukazatel odkazuje na odpovídající modul v původním slově. Tím je možné ztotožnit moduly kontextu pravidla obsahující pojmenovávací parametry s moduly původního slova s hodnotovými parametry. Toho je využito v dalším kroku.

Pokud je při aplikaci pravidla nutné vyhodnocovat výrazy (tzn. pravidlo obsahuje podmínku nebo formální parametry v následníkovi), následuje vytvoření nového prostředí pro tyto účely. Prostředím se rozumí objekt třídy `cparse::TokenMap`, který proměnným přiřazuje jejich hodnoty. Toto prostředí je lokální a jeho existence je omezena na právě probíhající aplikaci pravidla. Jeho předkem se stane globální prostředí, které je opět objektem `cparse::TokenMap`, ale je uložené v objektu `SystemConstants`, který je součástí objektu `SystemDefinition`.

Globální prostředí obsahuje hodnoty proměnných L-systému. Spojením pojmenovávacích a hodnotových parametrů kontextů pravidla a původního slova jsou proměnné se svými hodnotami přidané do lokálního prostředí. V lokálním prostředí se dále vyhodnotí podmínka pravidla a při jejím splnění i formální parametry následovníka pravidla. Vzniklé slovo s hodnotovými parametry je pak přidáno na konec nově vznikajícího slova.

5.5 Parsování definice L-systému

Třídy pro parsování textové definice L-systému jsou ve složce `src/parse/`. Ty, které jsou v podadresáři `src/parse/antlr4/`, jsou výstupem projektu nástroje ANTLR, který je umístěn v adresáři `src/antlr4 Grammar/` na přiloženém DVD.

Vstupním bodem pro parsování definice je metoda `parseSimulation` třídy `Parser`. Výstupem této metody je objekt typu `Simulation` s aktuálním slovem nastaveným na axiom L-systému. Na začátku parsování jsou vytvořeny prázdné objekty tříd `SystemDefinition` a `SystemConstants`. Do nich jsou v průběhu parsování přidávány přepisovací pravidla, respektive hodnoty proměnných. Nakonec je objekt `SystemConstants` předán objektu `SystemDefinition` a výsledný objekt `SystemDefinition` je předán konstruktoru třídy `Simulation`, čímž získáme výsledek parsování celé definice.

Parsování probíhá řádek po řádku. Nejprve jsou odstraněny komentáře a pomocí regulárních výrazů je rozhodnuto, jestli je řádek prázdný, obsahující přepisovací pravidlo, nebo přiřazení do proměnné. Použité regulární výrazy nejsou dostatečně silné, aby byly schopné rozpoznat celou strukturu pravidel a přiřazení. Slouží pouze k určení režimu, v jakém bude práce s řádkem probíhat.

Následuje zpracování řádku pomocí nástroje ANTLR. Obsahuje-li symboly, je potřeba je převést na jejich ID. K evidenci ID symbolů slouží předem vytvořený objekt třídy `SystemDefinition`. Do tohoto objektu je také předáno výsledné pravidlo, pokud ho řádek obsahuje. Pokud je na řádku přiřazení do proměnné, jsou údaje zapsány do připraveného objektu typu `SystemConstants`[17].

Během parsování definice je potřeba pracovat s výrazy. Ty jsou pomocí knihovny Cparse kompilovány a ukládány do vznikajících přepisovacích pravidel nebo rovnou vyhodnocovány. Vyhodnocování výrazů probíhá v globálním prostředí, které je uloženo v objektu `SystemConstants`. Do tohoto prostředí jsou také ukládány hodnoty číselných proměnných. Proto je možné při parsování vyhodnocovat výrazy obsahující proměnné definované na předchozích řádcích.

5.5.1 Gramatika nástroje ANTLR v4

Práce s nástrojem ANTLR probíhá z hlediska programu LSysVis tak, že je mu předán textový řetězec a jako výstup získáme derivační strom. Derivační strom

[17] S výjimkou proměnné `ignore`, jejíž hodnota se předává přímo objektu `SystemDefinition`.

je následně procházen pomocí objektu typu `LSysVisitor`, přičemž je vstup zpracován a výsledky jsou předány objektům `SystemDefinition` nebo `SystemConstants`.

Při zpracování řetězce ho *lexer* nejdříve rozdělí na *tokeny*. Z tokenů poté *parser* sestaví derivační strom podle dané gramatiky a daného výchozího pravidla. Pravidla lexeru a parseru jsou definovaná v externím projektu, v souborech `LSysLexer.g4` a `LSysParser.g4`.

Při čtení vstupního řetězce lexer generuje různé typy tokenů podle toho, v jakém režimu právě je. Výchozím režimem je režim parsování přepisovacích pravidel L-systému. V tomto režimu jsou vytvářeny např. tokeny s názvy modulů nebo tokeny pro znaky oddělující levý a pravý kontext, podmínku a levou a pravou stranu pravidla. Při přečtení znaku otevírací závorky se lexer přepne[18] do režimu `parameters`. Tento režim je určen pro výrazy, které mohou být parametry symbolů jakéhokoliv typu. Tento režim je také použit např. při zpracování výrazu podmínky nebo váhy pravidla. V tomto režimu nedochází k vytváření tokenů pro proměnné, literály atd., protože o práci s výrazy se stará knihovna Cparse. Místo toho jsou jen rozděleny jednotlivé výrazy oddělené čárkami. Je však potřeba pracovat se závorkami a dalšími znaky, které mohou znamenat potřebu ukončit tento režim a vrátit se do režimu předchozího. Posledním režimem lexeru je režim `assign`, který je lexeru explicitně nastaven před parsováním přiřazení do proměnné. Toto explicitní nastavení probíhá programově a nesouvisí s obsahem souboru `LSysLexer.g4`. Po vytvoření tokenu s názvem proměnné je lexer přepnut do výchozího režimu nebo do režimu `parameters` podle toho, o jaký typ proměnné jde.

Následuje převedení posloupnosti tokenů na derivační strom. To je provedeno za použití pravidel parseru. Voláním odpovídající metody parseru v programu je zvoleno, které z pravidel gramatiky má být použito pro celou posloupnost tokenů, což také určí, jaký bude kořen derivačního stromu. Při zpracování řádku s přepisovacím pravidlem to je pravidlo parseru `lSysRule`, u přiřazení do proměnné pravidlo `variableAssign`.

Nakonec je vytvořen objekt typu `LSysVisitor`. Jeho metodě odpovídající kořenu stromu je parserem vytvořený derivační strom předán. LSysVisitor strom projde, přičemž vytvoří vhodné datové struktury a uloží je do objektů `SystemDefinition` a `SystemConstants`.

5.6 Grafický výstup a želví grafika

Prací s L-systémem jsou vygenerována vykreslovací slova. Tato práce probíhá ve vláknu objektu `LSystemSimulationWorker` voláním metod objektu `Simulation`. Vykreslovací slovo je poté zpracováno želví grafikou ve vláknu objektu `LSystemRenderWorker`. Výstupem želví grafiky jsou data připravená pro zpra-

[18] Pro přepínání režimů slouží příkazy `mode`, `pushMode` a `popMode`. Více v dokumentaci nástroje: https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md#mode-pushmode-popmode-and-more

cování shadery OpenGL. Tato data jsou předána widgetu `RenderWidget`, který tvoří plátno a zajišťuje kreslení na plátno pomocí OpenGL. V této kapitole jsou popsány třídy související s želví grafikou a plátnem, které jsou umístěny v adresáři `src/render/`.

Hlavním zdrojem při implementování těchto tříd a při psaní této kapitoly byla kniha [4].

5.6.1 Třída `Turtle`

Tato třída zajišťuje veškerou funkcionalitu želví grafiky. Životnost objektu této třídy odpovídá jedné interpretaci slova L-systému. V konstruktoru je předán objekt `SystemConstants`, ze kterého je přečten počáteční stav želvy a inkrementační hodnoty pro interpretaci určitých symbolů. Poté je metodě `interpretWord` předáno vykreslovací slovo, čímž dojde k interpretaci modulů slova a vytvoření výstupních dat.

Stav želvy je reprezentován strukturou `TurtleState`. Objekty tohoto typu jsou při interpretaci symbolů `[` a `]` ukládány na zásobník a opět načítány. Pozice a orientace želvy jsou reprezentovány maticí o čtyřech řádcích a sloupcích. Při změně orientace nebo pozice želvy je na matici aplikována daná transformace. Datové typy a funkce pro práci s maticemi a vektory zajišťuje knihovna GLM. Uložení ostatních hodnot stavu želvy je přímočaré stejně jako interpretace symbolů, které mění tyto hodnoty.

Při provedení kroku vpřed a nakreslení čáry jsou údaje potřebné k vykreslení tohoto kroku uloženy. Definice struktury, která obsahuje všechny tyto údaje, je ve zdrojovém kódu 4. Matice na řádku 2 určuje pozici počátečního bodu a orientaci čáry. Dále je určena její barva, délka a tloušťka. Takto seskupená data jsou jednoduše vkládána do kontejneru `std::vector`. Na pořadí přitom nezáleží, protože z hlediska renderování nemají jednotlivé kroky želvy žádnou návaznost.

```
1 struct TurtleRenderState {
2     glm::mat4 position;
3     glm::vec3 color;
4     float length;
5     float thickness;
6 };
```

Zdrojový kód 4: Definice struktury `TurtleRenderState`.

5.6.2 Třída `RenderWidget`

Kontejner naplněný objekty `TurtleRenderState` je předán widgetu `RenderWidget` – tedy plátnu – k vykreslení. `RenderWidget` využívá API OpenGL verze 3.3 Core Profile, jehož funkce zpřístupňuje přímo framework Qt. `RenderWidget` navíc

dědí z třídy `QOpenGLWidget`, která zjednodušuje práci s OpenGL tím, že uživatele knihovny odstiňuje od inicializačních a dalších kroků, které přímo nesouvisí s vykreslovaným obsahem.

Funkce OpenGL je možné volat v metodách `initializeGL`, `resizeGL` a `paintGL`. Tyto chráněné metody je potřeba překrýt a implementovat v nich požadovanou funkcionalitu. Volání těchto metod je v režii knihovny, ale zavolání nejdůležitější z nich, metody `paintGL`, lze i vynutit pomocí metody `update`. To je užitečné např. při potřebě změnit vykreslované slovo na plátně. Knihovna zajišťuje, že při vykonávání těchto metod je aktivní správný kontext rozhraní OpenGL.

V metodě `initializeGL`, která je volána jen jednou po vytvoření widgetu, dojde k inicializaci a nastavení vykreslovacího rozhraní. Dále jsou zkompilovány shadery k renderování osového kříže a L-systému a je vytvořen *Vertex Array Object* (VAO) k renderování osového kříže.

Metoda `resizeGL` je zavolána, když dojde ke změně velikosti plátna. Její hlavní využití je, že informaci o změně velikosti předá dál do rozhraní OpenGL.

Metoda `paintGL` je volána pokaždé, když je potřeba překreslit plátno. Součástí této metody je také zpracování dat želví grafiky, pokud je to před překreslením potřeba udělat. Volání metody probíhá v několika krocích. Nejdříve je buffer plátna přepsán barvou pozadí. Poté, pokud je to potřeba, proběhne deinicializace starého VAO s daty želví grafiky a případné načtení nových dat. Nakonec je vykreslen osový kříž a data L-systému, pokud k tomu má dojít.

5.6.3 Třídy `BasicVAO` a `TurtleVAO`

Tyto třídy slouží ke správě VAO objektů. Při inicializaci VAO dojde k překopírování dat potřebných k vykreslení daného objektu z paměti programu do paměti, která je spravována knihovnou OpenGL. Často jde o paměť grafické karty. Poté, co je VAO inicializován, je možné ho pomocí příslušného shaderu vykreslit na plátno. Deinicializací pak dojde k uvolnění grafické paměti.

`BasicVAO` slouží k uložení libovolných bodů a jejich následnému vykreslení podle libovolného primitiva OpenGL. V programu je však využíván jen k vykreslení osového kříže.

`TurtleVAO` je o něco zajímavější. Při inicializaci jsou mu předána data želví grafiky a úroveň detailu. Na základě úrovně detailu jsou vygenerovány body, které při interpretaci primitivem `GL_TRIANGLE_STRIP` tvoří plášť pravidelného hranolu. Hodnota detailu určuje, z kolika obdélníkových stěn je plášť tvořen. Pro vyšší hodnoty detailu se může po vykreslení zdát, že jde o plášť válce. Souřadnice těchto bodů jsou předány rozhraní OpenGL.

Jeden takový útvar odpovídá při renderování jednomu kroku želvy. Údaje o tom, kde mají jednotlivé kroky želvy být a jaké mají další vlastnosti, jsou také předány rozhraní OpenGL.

Tím je dokončena inicializace objektu `TurtleVAO`. Inicializaci je potřeba provádět jen po přijetí nového výstupu želví grafiky. Pokud dojde např. ke změně

stavu kamery, inicializovaná data to nijak neovlivní. Kromě toho jsou také opakující se data, která popisují plášť hranolu, nahrána pouze jednou – k popisu jednotlivých kroků želvy jsou pak k dispozici jen ty údaje, které jsou skutečně potřeba. Tím je značně omezen objem dat, která je nutné přenášet do grafické paměti.

Vykreslení `TurtleVAO` proběhne zavoláním funkce `glDrawArraysInstanced`.

5.6.4 Třídy `BasicShader` a `TurtleShader`

Tyto třídy slouží ke správě shaderů. Shadery jsou programy, které jsou vykonávány grafickým procesorem a určují, jakým způsobem jsou data VAO zpracována a vykreslena. Zdrojové kódy shaderů jsou v adresáři `res/shaders/` projektu. V programu LSysVis jsou používány dva typy shaderů: vertex shader a fragment shader. Jejich spojením pak vznikne shader, který je schopný pracovat s objekty VAO dané struktury.

V metodě `initializeGL` jsou zkompilovány a načteny dva shadery: jeden pro `BasicVAO` a jeden pro `TurtleVAO`.

`BasicShader` není příliš komplikovaný. Jeho vertex shader umožňuje body z `BasicVAO` transformovat podle zadané matice a fragment shader umožňuje nastavit jejich barvu.

Ve vertex shaderu třídy `TurtleShader` dojde nejdříve k vytvoření škálovací matice, která změní souřadnice bodů pláště hranolu tak, aby jeho tvar odpovídal délce a šířce kroku. Po škálování jsou body pomocí translace a rotace umístěny na své místo v modelu L-systému. Nakonec je ještě aplikována matice přechodu vytvořená kamerou. Takto vzniklé body jsou předány dále a ve fragment shaderu dojde opět jen k nastavení barev.

5.6.5 Třída `Camera`

Účelem této třídy je uchovávat stav kamery, upravovat ho na základě vstupu od uživatele a generovat matice přechodu, které jsou poté použity v shaderech při vykreslování dat.

Objekt typu `Camera` je vlastněn widgetem `RenderWidget`. Události myši jsou zachyceny widgetem a předány objektu kamery. Stav kamery se změní na základě vstupu, ale také toho, jaký režim je kameře nastaven. Např. ve dvourozměrném režimu jsou některé hodnoty uzamknuty.

Stav kamery je dán souřadnicemi středového bodu, na který se kamera „dívá“, úhlem rotací kolem os X a Y a hodnotou přiblížení kamery ke středovému bodu. V objektu je také uložena hodnota poměru stran plátna, kterou je nutné aktualizovat při změně.

Z těchto hodnot a na základě aktuálního režimu kamery jsou generovány různé matice přechodu.

Zajímavá je hodnota `radius`, která udává skutečnou vzdálenost kamery od středového bodu. Tato hodnota je vypočítávána z hodnoty `zoom`, která se mění na základě vstupu od uživatele. K hodnotě `zoom` jsou přičítány a odečítány úhly

otočení kolečka myši. Může tedy nabývat kladných i záporných hodnot. V metodě `processZoomChanged` je z hodnoty `zoom` vypočítána skutečná vzdálenost kamery od středového bodu podle předpisu $radius = 1.001^{zoom}$. Tento způsob výpočtu přiblížení má několik výhod. Hodnota může být velmi malá i velká, ale vždy je kladná. Záporná vzdálenost od středového bodu by nedávala smysl. Díky exponenciálnímu růstu také platí, že když je objekt hodně oddálen, jsou skoky větší než když je přiblížen. To je užitečné, protože není problém objekt rychle oddálit ani není problém dělat detailní kroky při velkém přiblížení. Zároveň je zajištěno, že při pohybu kolečka dopředu a poté dozadu bude výsledná vzdálenost stejná, což je přirozené. Při změně hodnoty `zoom` je navíc upravena i hodnota `moveSpeed`, což zajišťuje, že proporcionálně ke změnám rychlosti přiblížení se mění i rychlost pohybu kamery.

5.7 Další vývoj programu

Program LSysVis se skládá z několika částí, přičemž každá z nich představuje prostor pro další vylepšování programu. Možnosti dalšího vývoje jsou tím pádem široké. V této kapitole je přehled funkcí a optimalizací, které by mohly být předmětem dalších kroků vývoje programu.

5.7.1 Rozšíření možností želví grafiky o kreslení mnohoúhelníků

Implementace želví grafiky v programu splňuje spíše jen základní požadavky pro vizualizaci L-systémů. V L-systémech jsou v souvislosti s želví grafikou často zaváděny symboly `{` a `}`, které umožňují vytvářet mnohoúhelníky. Princip, jakým je toho dosaženo, spočívá v tom, že symboly `F`, které jsou mezi těmito závorkami, nemají význam čar, ale vrcholů mnohoúhelníku. To s sebou přináší určité nároky: v L-systému je navíc potřeba kontrolovat vyváženost nového typu závorek a zároveň je nutné zajistit, že mezi těmito závorkami se želva pohybuje pouze v jedné rovině. Hlavní výzvou při implementaci této funkce je však to, že by bylo nutné rozšířit část programu starající se o renderování o zcela novou funkci vykreslování mnohoúhelníků. S tím jsou spojené problémy jak tyto plochy reprezentovat a jejich data předávat rozhraní OpenGL.

5.7.2 Vylepšení textového editoru

Textový editor v programu LSysVis je velmi jednoduchý. Bylo by na místě přidat funkci umožňující najít a případně nahradit řetězec v textu. Dalším vylepšením by mohlo být zvýrazňování syntaxu zadaných přepisovacích pravidel a přiřazení do proměnných. Framework Qt pro tyto účely poskytuje třídu `QSyntaxHighlighter`. Vzhledem k tomu, že v programu je už nyní k dispozici parser, který syntaxu rozumí, nemuselo by přidání této funkce být příliš složité.

5.7.3 Optimalizace práce s L-systémy

Práce s jednoduchými L-systémy je v programu relativně rychlá. Ve většině případů je omezující spíše kapacita operační paměti něž výpočetní výkon a není problém pracovat se slovy obsahujícími miliony symbolů. Situace je jiná u parametrických L-systémů. Už při dosažení velikosti slova v řádech desítek tisíc symbolů dochází ke znatelnému zpomalení. Profilováním běhu programu pomocí nástroje *Very Sleepy*[19] se ukázalo, že hlavní příčinou zpomalení je volání funkce `cparse::calculator::eval`, která slouží k vyhodnocení výrazů. Volání této funkce se bohužel nedá jednoduše obejít. Řešením by tedy byl přechod na jinou knihovnu pro vyhodnocování výrazů. Vhodným kandidátem by mohla být knihovna *ExprTk*[20], která slibuje nejen zlepšení výkonu, ale i robustnější možnosti konfigurace a odhalení některých chyb už při kompilaci výrazu – např. že je použita proměnná, které předtím nebyla přiřazena hodnota. Tyto chyby je v současnosti možné odhalit až při vyhodnocování výrazu.

Generování slov by bylo také možné paralelizovat. Jednou možností by bylo generovat další aktuální slovo současně se slovem vykreslovacím. Zajímavější možností by však mohlo být rozdělení každého derivovaného slova na tolik částí, kolik má počítač k dispozici vláken výpočtu. Jednotlivé části slova by byly přepsány zvlášť a poté spojeny do výsledného slova. Použitím tohoto přístupu by byla využita celá výpočetní síla procesoru. Na druhou stranu u jednodušších L-systémů není aplikace pravidel příliš výpočetně náročná, ale vyžaduje spíše překopírování určitého množství dat. Pokud by bylo slovo rozděleno, jeho spojení do výsledné podoby by naopak přidalo objem dat, které by bylo potřeba kopírovat.

5.7.4 Zavedení speciální proměnné `seed`

U stochastických L-systémů dojde při každém načtení nebo resetování simulace k vytvoření nového semínka pro generátor pseudonáhodných čísel. Hodnota tohoto semínka (číselná hodnota) by mohla být zobrazena uživateli. Speciální proměnné `seed` by pak mohla být tato hodnota přiřazena a tím by bylo zaručeno, že stochastický L-systém by po provedení určitého počtu derivačních kroků byl ve stejném stavu jako při původní derivaci. Implementace této funkce by v současnosti nebyla příliš složitá, ale pokud by došlo k přechodu na paralelní aplikování přepisovacích pravidel, jak je popsáno výše, implementace této funkce by byla komplikovanější.

[19] http://www.codersnotes.com/sleepy/
[20] http://www.partow.net/programming/exprtk/

Závěr

V rámci této práce vznikl program LSysVis, který umožňuje definovat L-systémy, odvozovat jejich derivační kroky a nechat si zobrazit výsledky v grafické formě. Pro vykreslení výsledků byla vytvořena vlastní implementace želví grafiky.

Jedním z požadavků na program byla co nejširší podpora různých typů L-systémů. Tento cíl se podařilo do značné míry splnit. Typy L-systémů, se kterými program pracuje, jsou představeny v kapitole 2.

Pro zadání definic L-systémů byl vytvořen jazyk, který vychází z formálního způsobu zápisu L-systémů. Uživatelské rozhraní programu bylo navrženo tak, aby bylo jednoduché a dovolilo uživateli experimentovat s L-systémy bez zbytečných překážek. Dokumentace jazyka a uživatelského rozhraní programu je v kapitole 4.

Možnosti želví grafiky v programu nejsou příliš bohaté, ale dostačující k základní práci s L-systémy. Silnými stránkami implementace želví grafiky jsou dobrá podpora práce v dvourozměrném i trojrozměrném prostoru, plynulé přepínání mezi těmito režimy a dobrá optimalizace.

Jedním z nedostatků želví grafiky v programu LSysVis je chybějící podpora kreslení mnohoúhelníků. O této a o dalších oblastech, ve kterých je možný další vývoj programu, pojednává kapitola 5.7.

Conclusions

The result of this thesis is a desktop program called LSysVis. It lets its users define L-systems, simulate their derivation steps and render the results in a graphical form. For this purpose, an original implementation of turtle graphics was developed.

One of the requirements was to support a wider range of different types of L-systems. This goal has been largely met. Supported types of L-systems are presented in chapter 2.

For the purpose of describing L-systems, a new language has been developed. This language is based on the way in which L-systems are formally written down. The user interface of LSysVis was designed to be as simple as possible and to allow users to experiment with L-systems without unnecessary obstacles. The documentation of the language and the user interface is in chapter 4.

The features of the turtle graphics implementation are relatively limited but sufficient to enable basic work with L-systems. The turtle graphics implementation's strengths are good support for both two- and three-dimensional rendering modes and switching between them as well as good optimization.

One of the shortcomings of LSysVis is the lack of support for rendering polygons with turtle graphics. This and other areas where further development is possible are discussed in chapter 5.7.

A Obsah přiloženého DVD

bin/
: Spustitelný soubor programu LSysVis a další soubory potřebné pro jeho provoz.

data/
: Ukázkové soubory definic L-systémů ve formátu programu LSysVis.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

install/
: Instalátor runtime knihovny potřebné pro provoz programu LSysVis.

src/
: Kompletní zdrojové texty programu LSysVis se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu.

readme.txt
: Instrukce pro instalaci a spuštění programu LSysVis, včetně všech požadavků pro jeho bezproblémový provoz.

Literatura

[1] PRUSINKIEWICZ, Przemyslaw. Graphical applications of l-systems. In. *Proceedings of Graphics Interface '86 — Vision Interface '86*. 1986, s. 247–253.

[2] LINDENMAYER, Aristid; PRUSINKIEWICZ, Przemyslaw. *The algorithmic beauty of plants*. New York: Springer-Verlag, 1990. ISBN 9781461384762.

[3] PRUSINKIEWICZ, Przemyslaw; HAMMELY, Mark; HANANZ, Jim; MĚCH, Radomír. *L-systems: from the Theory to Visual Models of Plants*. Alberta, Canada: CSIRO Publishing, 1996.

[4] DE VRIES, Joey. *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*. Netherlands: Kendall & Wells, 2020.