VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY GENEROVÁNÍ ROSTLIN PRO POČÍTAČOVOU GRAFIKU

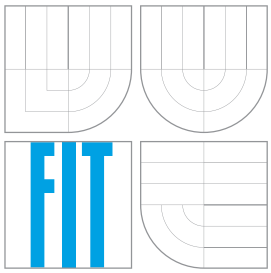
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ PICEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY GENEROVÁNÍ ROSTLIN PRO POČÍTAČOVOU GRAFIKU

METHODS OF GENERATING PLANTS FOR COMPUTER GRAPHICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ PICEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MILET

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá metodami využitelnými pro generování rostlin pro počítačovou grafiku. Nejpodrobněji je zde probraná metoda generování pomocí L-systémů. Práce popisuje jednotlivé rozšíření 0L-systémů. Pro získání geometrické reprezentace modelu je výstup L-systému zpracován želví grafikou. Geometrická reprezentace modelu je následně pomocí grafické knihovny OpenGL vykreslena. Textury jsou generovány Perlinovou šumovou funkcí implementovanou v jazyce GLSL.

Abstract

This Bachelor's thesis deals with methods useable for a plant generation for computer graphics. Most detailed described method is generation by L-system. The thesis describes idividually extensions of 0L-systems. To get a geometric representation of model, the output of the L-system is proceed by turtle graphics. The geometric representation of model is drawn using OpenGL graphics library. Textures are generated by Perlin's noise function implemented in GLSL.

Klíčová slova

Procedurální generování, fraktální geometrie, L-systémy, modelování rostlin, fraktály, želví grafika, Perlinův šum

Keywords

Procedural generating, fractal geometry, L-system, plant modeling, fractals, turtle graphics, Perlin noise

Citace

Jiří Picek: Metody generování rostlin pro počítačovou grafiku, bakalářská práce, Brno, FIT VUT v Brně, 2015

Metody generování rostlin pro počítačovou grafiku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta.

.....
Jiří Picek
19. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce panu Ing. Tomášovi Miletovi za poskytnutí odborné pomoci.

Obsah

Kapitola 1

Úvod

V současné době se počítačová grafika využívá v počítačových hrách, ve filmech a dalších oblastech. Je snaha vymodelovat objekty co nejpodobnější realitě. Počítačový model je mnohdy levnější a rychleji vytvořený než reálný objekt. V dnešní době je mnoho způsobů, jak toho dosáhnout. Je možnost naskenování modelu nebo usnadnit modelování využitím softwaru pro trojrozměrnou grafiku. Takto lze vymodelovat lidmi vytvořené objekty. Jsou zde ale omezení. Pokud má být objektů mnoho, načítání z pevného disku trvá dlouho. V počítači je trvalé uložiště s největší kapacitou, proto je vhodné pro uložení velkých modelů, ale je také nejpomalejší, proto trvá načítání dlouho a aplikace musí čekat na načtení požadovaných dat. Další omezení je, pokud chceme model objektu, který vytvořila příroda jako například strom. Objekt je příliš členitý, proto ho nelze jednoduše naskenovat.

Je tedy nutné použít jinou metodu. Jedna z možností je naklikat bod po bodu, jak bude výsledný model vypadat, ale to je zdlouhavé. Výstup je kvalitnější než při předchozí metodě, ale s více detaily se zvýší i nároky na paměť pro uložení. V dnešní době, kdy se začínají čím dál více prosazovat cloudová uložiště, je dobré minimalizovat výsledný model, ale tak, aby nepřišel o detaily. Je tedy možnost použít metodu procedurálního modelování.

Tato bakalářská práce se zabývá metodou procedurálního modelování rostlin. Metoda procedurálního modelování popisuje model algoritmem. Samotný model vzniká až za běhu aplikace. Algoritmus má téměř zanedbatelnou velikost oproti finálnímu modelu. Načtení z pevného disku nebo ze sítě je tedy rychlé. Většinou algoritmy bývají iterativní, proto je možné vygenerovat modely s různou úrovní detailů. Pro vzdálené objekty není potřeba tolik detailů, jako když je objekt blízko. Různé objekty lze generovat pomocí fraktální geometrie popsané v kapitole 2.1.1. Do fraktální geometrie je možné zahrnout i L-systémy.

V této práci je implementováno generování rostlin pomocí L-systémů, které byly vyvinuty pro simulaci růstu rostlin maďarským biologem Aristidem Lindenmayerem. Tento systém je založen na přepisování řetězce podle zadaných přepisovacích pravidel. Pokročilé L-systémy umožňují generování realistických modelů rostlin. Lze s nimi generovat dokonce celé rostlinné systémy, které reagují na své okolí. L-systémy lze využít i pro generování jiných objektu než jsou rostliny, ale to je již mimo rozsah této práce. Popis L-systémů je v kapitole 2.1.2.

Kapitola 2

Metody generování

Trojrozměrný model objektu je možné vytvořit různými postupy a metodami. Pokud máme k dispozici reálný model a scanovací zařízení, můžeme objekt naskenovat. Skenování je možné provést pomocí prostorového skeneru, sonaru nebo jiného prostorového snímacího zařízení. Záleží na tom, zda potřebujeme model s vnitřními informacemi. Většinou si vystačíme s povrchovou reprezentací, kde nejsou obsaženy informace o vnitřní struktuře. Pokud nemáme k dispozici prostorové scanování, je možné nasnímat objekt fotoaparátem či kamerou. Následně pomocí počítačového vidění převést fotografie či video na počítačový model [7]. Tomuto způsobu se říká *na obrazech založené modelování (image based modeling)*. Tento způsob je časově nenáročný a jednoduché modely vytvořené touto metodou mohou vypadat téměř jako modelovaný objekt. Problém nastává u členitých objektů, jako je například hlava s vlasy nebo strom s listy, případně dalšími detaily pro další simulaci. Obvykle je požadavkem odlišit jednotlivé vlasy na hlavě a jednotlivé listy na stromě. Počítačový model takového objektu může být deformovaný, nebo málo detailní.

Dalším způsobem je *interaktivní modelování*. Grafik vytvoří za pomoci klávesnice, myši či jiného vstupního zařízení počítačový model. Takové modely mohou být přesné, přičemž nezáleží na velikosti či dostupnosti objektu. Objekt není závislý na realitě, je omezen pouze představivostí grafika. Tento způsob je časově náročný a vyžaduje zkušenosti a talent. Zjednodušení modelování je možné pomocí 3D modelovacího softwaru. Je v něm možné využít hierarchického návrhu. To znamená, že se pracuje s jednoduchými objekty a pomocí dostupných nástrojů se slučují a upravují do složitějších objektů. Proto se metoda někdy nazývá *poloautomatická*.

V této práci je však nejvíce popsaná metoda procedurálního modelování. Model je reprezentován nějakým kódem či algoritmem [7]. Vlastností procedurálních technik je jednoduchost popisu modelu. Tuto metodu lze rozdělit na dvě základní skupiny. První jsou metody používané v CAD a CAGD jako například generování ploch z křivek. Druhá skupina se zaměřuje na automatické generování objektů, které se vizuálně nebo chováním podobají objektům, se kterými se setkáváme v reálném životě, v přírodě. Je tedy možné vymodelovat geometricky složité tvary. Dále se v této práci budu zabývat pouze druhou skupinou procedurálního modelování.

2.1 Procedurální generování

Metody procedurálního generování je možné rozdělit do několika skupin, kterými jsou fraktální geometrie, Perlinův šum, L-systémy a další. Existuje mnohem více skupin, do kterých

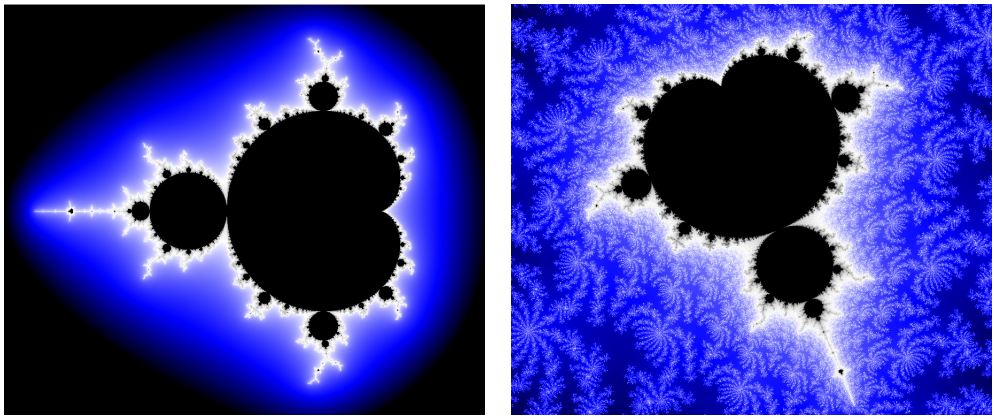
lze rozdělit procedurální generování, ale pro tuto práci jsou důležité pouze tyto tři zmíněné metody. Pomocí těchto metod je možné popsat reálné objekty algoritmicky. Celý model se podle zapsaného algoritmu vygeneruje až za běhu aplikace, nároky na paměť pro uložení jsou tedy téměř nulové. Ve většině případů bude model vygenerovaný rychleji, než kdyby se načítal ze souboru po jednotlivých bodech z trvalého uložiště[1]. Trvalé uložiště je technologicky nejdále od procesoru, je to tedy jedna z nejpomalejších pamětí v počítači.

Procedurální generování není výhodné jenom pro modely, procedurálně lze vytvořit i textury. Taková textura se na celém objektu nemusí opakovat, přesto je její tvorba rychlá. Výhody jsou zde stejné jako při generování modelů.

2.1.1 Fraktální geometrie

Na přelomu devatenáctého a dvacátého století se začaly objevovat matematické konstrukce objektů, které se lišily od Euklidovské geometrie [6]. Jedna z prvních takových konstrukcí neměla v žádném bodě derivaci. Byla objevena Karlem Weierstrassem už v roce 1872. Helge von Koch přišel v roce 1904 s nekonečně dlouhou křivkou, která ohraničuje konečnou plochu. Takovéto konstrukce byly mnohými matematiky odmítány, později se však ukázalo, že popisují přírodní jevy lépe než útvary klasické geometrie. V roce 1919 vytvořil Felix Hausdorff definici fraktální dimenze, která pro mnohé objekty vychází neceločíselně a v některých případech dokonce iracionálně.

Fraktální geometrie je samostatná rozsáhlá vědní disciplína [7], která vznikla a je rozvíjena od šedesátých let dvacátého století. Za objevitele fraktálu se považuje polský vědec Benoit B. Mandelbrot. Jako první definoval pojem fraktál. Jedním z nejznámějších fraktálních útvarů je Mandelbrotova množina, což je dynamický fraktál ležící v komplexní rovině, ukázka je na obrázku 2.1 vlevo.



Obrázek 2.1: Mandelbrotova množina – celá (vlevo) a výřez (vpravo). Vygenerováno pomocí dynamické plochy v grafickém prostředí KDE 4.5

Objekty se dají rozdělit do dvou skupin. Jsou to geometricky hladké objekty a nekonečně členité objekty. Pokud vezmeme geometricky hladkou křivku, která má topologickou dimenzi rovnou jedné, při měření dostaneme vždy stejnou délku bez ohledu na velikost měřidla. Pokud ale budeme měřit délku pobřeží, což je také křivka s topologickou dimenzí

[1] Tralé uložiště je paměť ve které zůstávají data i po vypnutí počítače. Může to být pevný disk, CD, disketa, flash disk.

rovnou jedné, při postupném zmenšování měřidla bude délka větší, postupně až nekonečně velká. Pobřeží tedy zabírá více místa v rovině než geometricky hladká křivka. Nezabírá však celou plochu. Dimenze je větší než jedna, ale menší než dvě, což je v rozporu s topologickou dimenzí, která je vždy celočíselná. Tato neceločíselná dimenze se nazývá *fraktální dimenze*.

Empirický vztah mezi délkou a velikostí měřidla je definovaný vztahem $K = C\varepsilon^D$, kde $\varepsilon > 0$ je délka měřidla, C je konstanta úměrnosti, K je celková délka aproximace měřeného útvaru a D je fraktální dimenze, která nemusí být celočíselná.

Všeobecně platná definice fraktálu prozatím neexistuje. Jedna z možných definic fraktálu je, že fraktál je objekt, jehož geometrický motiv se opakuje v něm samém až do nekonečna [6]. Podle chápání nekonečna lze fraktály rozdělit do dvou skupin. Pokud bude bráno nekonečno v matematickém smyslu, je to skupina soběpodobných fraktálů. Soběpodobné fraktály jsou pouze teoretické a existují jenom v matematických konstrukcích. Jejich specifikem je, že se v nich opakuje původní motiv základního útvaru či tělesa. Příkladem může být Mandelbrotova množina, viz obrázek 2.1 vpravo. Druhá skupina je založena na nekonečnu z fyzikálního světa. Fraktály v této skupině se nazývají soběpříbuzné. Setkáváme se s nimi každý den, aniž bychom si to uvědomovali. Příkladem takovýchto fraktálů jsou mraky, lesy, hory, vodní hladina, ale i objekty jako obličej. Tyto fraktály mají své limity. Pro vysvětlení vezměme strom. Když se z něj část vyřízne, výřez bude podobný původnímu stromu. Výřezy je možné provádět až do doby, kdy zbyde nejmenší větvička, za níž už není žádná fraktální struktura. Toto je fyzikální hranice systému.

2.1.2 L-systémy

L-systémy vytvořil v šedesátých letech maďarský biolog Aristid Lindenmayer. Jsou na jeho počest pojmenovány jako Lindenmayerovy systémy zkráceně jako L-systémy. Byly koncipovány jako matematická teorie vývoje rostlin. Jsou založené na systému přepisovacích pravidel. Původně neobsahovaly dostatek informací pro komplexní modelování vyšších rostlin. Hlavní byla topologie rostlin, což je například vztah mezi sousedními buňkami nebo většími rostlinými částmi. Jejich geometrický vzhled nebyl důležitý. Následně bylo několik geometrických interpretací navrženo s vyhlídkou na všestraný nástroj pro modelování rostlin. Pro převod do geometrického popisu se v této práci používá želví grafika.

Želví grafika

Želví grafika je populární forma představování programování dětem. Byla to část programovacího jazyka Logo, vytvořeného Wallym Feuzigem a Seymourem Papetem v roce 1966 [1].

Základem želví grafiky je virtuální želva, která se řídí pomocí příkazů. V této práci jsou příkazy vygenerované L-systémem, který je probrán podrobněji dále.

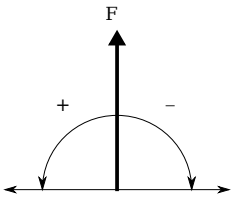
Želva je definována svým stavem a množinou příkazů. Stav je reprezentován její aktuální polohou a orientací. V případě dvourozměrného prostoru je stav popsán bodem a úhlem natočení. Množina příkazů, které je želva schopna provádět, je F, f, $+$, $-$. Jejich význam je popsán v tabulce 2.1. Želva začíná v bodě $[0, 0, 0]$ a směřuje vzhůru – úhel je 90°, viz obrázek 2.2.

Nyní víme, jak pomocí želví grafiky vykreslit dvourozměrné objekty. Ale reálný svět tak jak ho známe je trojrozměrný a rostliny v něm také. Je potřeba rozšířit dvourozměrnou implementaci do trojrozměrné. Množina příkazů, o které se rozšířila původní, je $\wedge$, &, /, \, |. Jejich význam je popsán v tabulce 2.2.

Orientace želvy již nemůže být reprezentována pouze jedním úhlem. V trojrozměrném prostoru je orientace želvy reprezentována trojicí vektorů $\vec{H}$ (heading), $\vec{L}$ (left) a $\vec{U}$ (up).

$F(s)$	Posun vpřed o délku s a vykreslení čáry mezi původní a novou pozicí želvy.
$f(s)$	Posun vpřed o délku s bez vykreslení čáry.
$+(\alpha)$	Otočení želvy vlevo o úhel α.
$-(\alpha)$	Otočení želvy vpravo o úhel α.

Tabulka 2.1: Příkazy, které 2D želva zpracovává



Obrázek 2.2: Počáteční nastavení 2D želvy a znázornění funkce příkazů, převzato z [4, str. 7]

Tyto vektory jsou jednotkové, navzájem na sebe kolmé a musí splňovat rovnici 2.1. Spolu s polohou vytvářejí lokální souřadnicový systém. Jsou zobrazeny na obrázku 2.3.

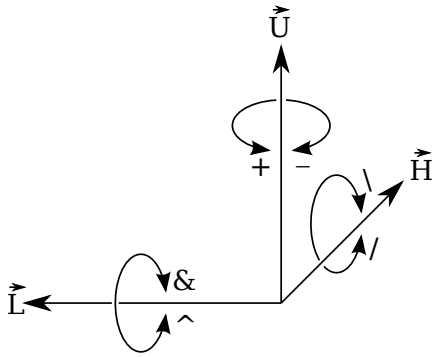
$$\vec{H} \times \vec{L} = \vec{U} \tag{2.1}$$

Rotace želvy se řídí rovnicí 2.2, kde R je matice 3×3. Rotace podle jednotlivých vektorů mohou být popsány rotačními maticemi 2.3, 2.4 a 2.5.

Změny stavu želvy jsou způsobeny interpretací symbolů, z nichž každý může být následován parametry. Pokud existuje více než jeden parametr, první vždy ovlivňuje stav želvy. Pokud nenásledoval žádný parametr, je použita defaultní hodnota nastavená mimo želvu [3].

$+(\alpha)$	Otočení želvy vlevo o úhel α okolo vektoru $\vec{U}$.
$-(\alpha)$	Otočení želvy vpravo o úhel α okolo vektoru $\vec{U}$.
$\&(\alpha)$	Náklon želvy dolů o úhel α okolo vektoru $\vec{L}$.
$\wedge(\alpha)$	Náklon želvy nahoru o úhel α okolo vektoru $\vec{L}$.
$/(\alpha)$	Převalení želvy vlevo o úhel α okolo vektoru $\vec{H}$.
$\backslash(\alpha)$	Převalení želvy vpravo o úhel α okolo vektoru $\vec{H}$.
$\vert$	Otočení želvy o 180° okolo vektoru $\vec{U}$.

Tabulka 2.2: Rozšíření příkazů pro 3D želvu



Obrázek 2.3: Rotační vektory želvy a znázornění funkce rotačních příkazů, převzato z [3, str. 10]

$$\left[\vec{H}'\vec{L}'\vec{U}'\right] = \left[\vec{H}\vec{L}\vec{U}\right]\mathbf{R} \tag{2.2}$$

$$\mathbf{R_U}(\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$\mathbf{R_L}(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \tag{2.4}$$

$$\mathbf{R_H}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{2.5}$$

Přepisovací pravidla

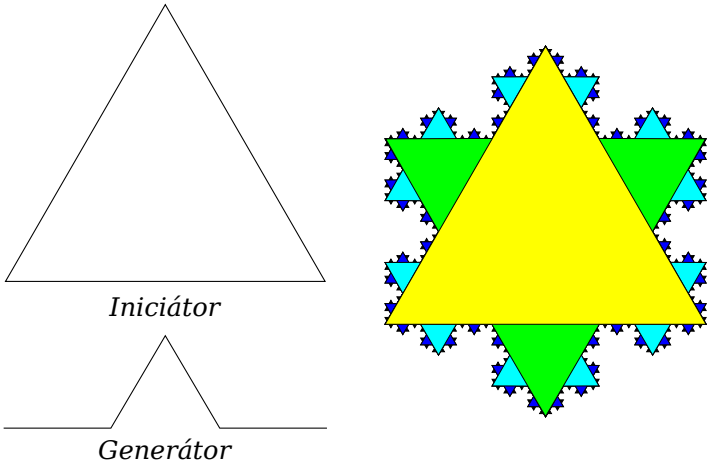
Základ L-systémů je založen na přepisování [4]. Přepisování obecně je technika generování složitých objektů postupným nahrazováním částí jednoduchého počátečního objektu. Používá se k tomu seznam pravidel.

Jedním z nejznámějších objektů, který používá přepisovací pravidla je Kochova vločka. Počáteční obrazec, nazývaný iniciátor, je u Kochovy vločky trojúhelník. Generátor, kterým se dále jednotlivé strany nahrazují, je křivka. Nejznámější generátor pro Kochovu vločku je úsečka rozdělená na třetiny a prostřední část je nahrazená trojúhelníkem bez spodní strany. Na obrázku 2.4 je zobrazen iniciátor, generátor a několik prvních iterací.

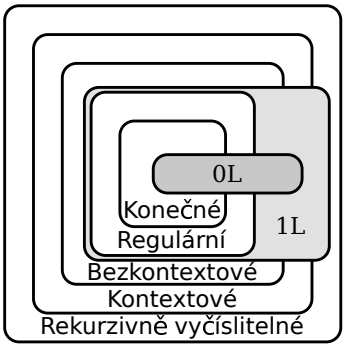
Gramatika L-systémů se liší rozgenerováním od Chromského gramatik. U Chromského gramatik jsou aplikována pravidla sekvenčně, ale u L-systémů se aplikují paralelně. To ovlivňuje možnosti vygenerovaných jazyků. Jazyky, které lze vygenerovat bezkontextovým L-systémem, nemusí jít vygenerovat bezkontextovou Chromského gramatikou. Vztah mezi třídami Chromského gramatik a L-systémů je zobrazen na obrázku 2.5.

Deterministické bezkontextové L-Systémy

Deterministický bezkontextový L-systém, často bývá označován jako D0L-Systém, jeho definice je popsána níže.



Obrázek 2.4: Kochova vločka – Iniciátor a generátor (vlevo), Barevně odlišené iterace generování vločky. Od iniciátoru po čtvrtou iteraci jsou barvy: žlutá, zelená, zelenomodrá, modrá, černá (vpravo)



Obrázek 2.5: Vztah mezi Chromského gramatikami a L-systémy. Symboly 0L a 1L značí bezkontextové a kontextové L-systémy, převzato z [4, str. 3]

Nechť V je abeceda, V^* je pak množina všech slov nad abecedou V a V^+ je množina všech neprázdných slov nad abecedou V. *Řetězec 0L-systému* je uspořádaná trojice $G = \langle V, \omega, P \rangle$, kde:

- V je *abeceda* systému,
- $\omega \in V^+$ je neprázdné slovo zvané *axiom*,
- $P \subset V \times V^*$ je konečná množina přepisovacích pravidel.

Přepisovací pravidlo $(a, \chi) \in P$ je zapsáno jako $a \to \chi$. Písmeno a a slovo χ jsou zvány jako *předchůdce* a *následovník* tohoto přepisovacího pravidla. Je předpokládáno, že pro jakékoliv písmeno $a \in V$, je zde alespoň jedno slovo $\chi \in V^*$ jako $a \to \chi$. Pokud žádné přepisovací pravidlo není explicitně specifikováno pro daného předchůdce $a \in V$, *přepisovací pravidlo identity* $a \to a$ se předpokládá, že patří do množiny přepisovacích pravidel P. 0L-systém je *deterministický* pouze a jedině, když pro každé $a \in V$ je zde přesně jedno $\chi \in V^*$ jako $a \to \chi$ [4, str. 4].

Jedná se o nejjednodušší typ L-systémů. Písmeno D v názvu označuje determinismus, to znamená, že pro každý symbol řetězce existuje maximálně jedno pravidlo. Nula znamená, že se jedná o bezkontextový přepis [7].

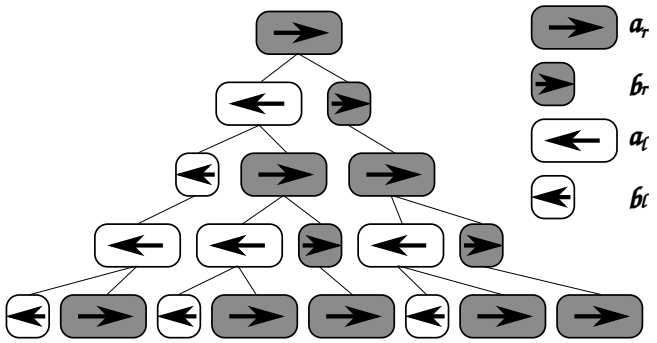
S těmito L-systémy lze simulovat vývoj mnohobuněčných struktur. Příkladem může být vývoj bakterie *Anabaena catenula*. Vývoj je popsán L-systémem 2.6, kde znaky a a b reprezentují stádia buněk (jejich velikost a připravenost na dělení). Indexy l a r určují polaritu buňky.

$$\begin{aligned} V &: \langle a_l, a_r, b_l, b_r \rangle \\ \omega &: a_r \\ p_1 &: a_r \rightarrow a_l b_r \\ p_2 &: a_l \rightarrow b_l a_r \\ p_3 &: b_r \rightarrow a_r \\ p_4 &: b_l \rightarrow a_l \end{aligned} \tag{2.6}$$

Když se začnou aplikovat přepisovací pravidla, dostaneme následující řetězce:

$$\begin{aligned} &a_r \\ &a_l b_r \\ &b_l a_r a_r \\ &a_l a_l b_r a_l b_r \\ &\dots \end{aligned}$$

Názornější je však grafická interpretace na obrázku 2.6. Pomocí L-systémů lze simulovat pouze diskrétně, proto není v tomto modelu zachycen postupný růst buněk mezi děleními.

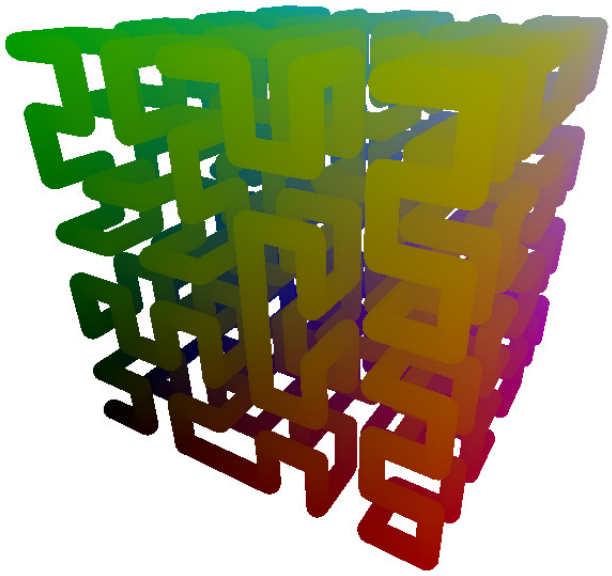


Obrázek 2.6: Vývoj bakterie Anabaena Catenula simulovaný D0L-systémem

L-systémy lze rozdělit na tři kategorie podle typu nahrazování. Jsou jimi:

- Hrany nahrazující
- Vrcholy nahrazující
- Hrany i vrcholy nahrazující

Jak názvy napovídají, nahrazují se buď hrany nebo se nahrazují vrcholy a hrany zůstávají. V práci bude využita Hilbertova křivka. Patří do vrcholů nahrazujících L-systémů. Vzniklá křivka postupně vyplňuje celý prostor, nikdy se neprotne, je soběpodobná a jednoduchá. Trojrozměrná Hilbertova křivka je zobrazena na obrázku 2.7 a lze popsat L-systémem 2.7.



Obrázek 2.7: Trojrozměrná Hilbertova křivka

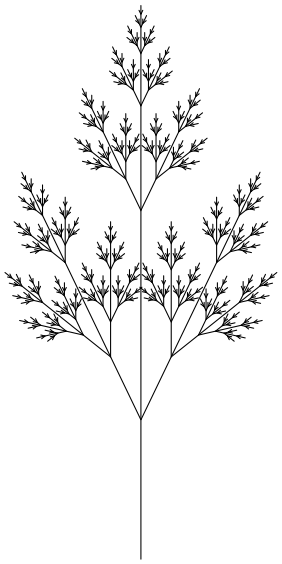
$$
\begin{aligned}
&n = 2, \delta = 90^\circ \\
&\omega = A \\
&A \rightarrow B - F + CFC + F - D\&F \wedge D - F + \&\&CFC + F + B// \\
&B \rightarrow A\&F \wedge CFB \wedge F \wedge D \wedge \wedge - F - D\wedge \mid F \wedge B \mid FC \wedge F \wedge A// \\
&C \rightarrow \mid D\wedge \mid F \wedge B - F + C \wedge F \wedge A\&\&FA\&F \wedge C + F + B \wedge F \wedge D// \\
&D \rightarrow \mid CFB - F + B \mid FA\&F \wedge A\&\&FB - F + B \mid FC//
\end{aligned}
\tag{2.7}
$$

Prozatím představené L-systémy se nevětví. Pokud je rostlina jednoduchá, která má pouze hlavní stonek a rozvětvení tvoří pouze listy, je možné takovou rostlinu zapsat přidáním nového významu dalším symbolům. Pokud přidáme symbolu L význam vykreslení listu do strany, je možnost získat rostlinu. Tímto způsobem však většinu rostlin vygenerovat nemůžeme, protože se větví.

Závorkové 0L-systémy

Pro získání rozvětvených rostlin je potřeba přidat novou funkčnost. Touto funkčností je možnost uložit stav želvy na zásobník a potom ho znovu načíst. Symbol pro uložení stavu na zásobník je znak „[“a pro načtení ze zásobníku je znak „]“.

Pokud se tyto symboly přidají do interpretovaného řetězce, je možné generovat již rozvětvené struktury. Příklad takové struktury je na obrázku 2.8, který je vygenerovaný podle L-systému 2.8.



Obrázek 2.8: Příklad závorkového L-systému, převzato z [4, str.25]

$$
\begin{aligned}
&n = 7, \delta = 25.7^\circ \\
&\omega = X \\
&X \to F[+X][-X]FX \\
&F \to FF
\end{aligned}
\tag{2.8}
$$

Parametrické L-Systémy

Parametrický L-systém rozšiřuje základní koncept L-systémů o možnost přidat číselné atributy k symbolům L-systému [3]. Pracuje se slovy patřícími do abecedy V s přiřazenými parametry patřícími do množiny reálných čísel $\Re$. Slovo $A \in V$ a jeho parametry $a_1, a_2, \ldots, a_n \in \Re$ jsou zapisovány jako $A(a_1, a_2, \ldots, a_n)$. Tímto způsobem jsou zapsány i příkazy želvy v tabulkách 2.1 a 2.2.

Parametrický 0L-systém je definovaný jako uspořádaná čtveřice $G = \langle V, \Sigma, \omega, P \rangle$, kde:

- V je abeceda systému,
- Σ je množina formálních parametrů,
- ω je neprázdné slovo zvané axiom,
- P je konečná množina přepisovacích pravidel.

Můžeme tedy ovlivňovat výsledek vygenerovaného modelu. Parametry mohou ovlivňovat velikost úhlu natočení, délku a tloušťku segmentu a tak dále. S touto možností ovlivňování lze vygenerovat ještě věrnější modely rostlin než s předchozími L-systémy.

Příklad zápisu pravidla parametrického L-systému je zapsán ve vztahu 2.9, kde $A(t)$ je předchůdce, $t > 5$ je podmínka a $B(t+1)CD(t \wedge 0.5, t-2)$ je následovník. V příkladu je

vidět, že nepřibyly samotné parametry za symboly L-systému, ale objevila se zde i podmínka oddělená znakem : od předchůdce. Pravidlo se aplikuje jenom tehdy, je-li podmínka splněna.

$$A(t) : t > 5 \rightarrow B(t+1)CD(t \wedge 0.5, t-2) \tag{2.9}$$

Parametr t má pro celé pravidlo stejnou hodnotu. Lze si ho představit jako proměnnou, kterou má definovanou předchůdce a předá jí následovníkovi. Výrazy se před předáním následovníkovi vyhodnotí. V axiomu se v parametrech mohou vyskytovat pouze čísla. Příklad celé definice je uveden v L-systému 2.10.

$$\begin{aligned} \omega &: B(2)A(4,4) \\ A(x,y) &: y <= 3 \rightarrow A(x*2, x+y) \\ A(y,y) &: y > 3 \rightarrow B(x)A(x/y, 0) \qquad (2.10) \\ B(x) &: x < 1 \rightarrow C \\ B(x) &: x >= 1 \rightarrow B(x-1) \qquad (2.11) \end{aligned}$$

Stochastické L-systémy

Všechny rostliny vygenerované deterministickým L-systémem jsou shodné. Pokud bychom použili více rostlin vygenerovaných stejným L-systémem ve stejné scéně, vypadalo by to uměle. Aby se tomu předešlo, je potřeba zanést do L-systému prvek náhody. Pokud se poruší pravidlo, že pro každý symbol existuje nejvýše jedno pravidlo, L-systém ztratí deterministické chování. Výběr pravidla potom závisí na pravděpodobnosti. Příkladem může být L-systém 5.3[2]. Pravděpodobnost, s jakou bude pravidlo vybráno, se zapisuje nad derivační symbol šipky. Součet všech pravděpodobností musí být jedna.

$$\begin{aligned} &\omega = F \\ &F \xrightarrow{.33} F[+F]F[-F]F \qquad (2.12) \\ &F \xrightarrow{.33} F[+F]F \\ &F \xrightarrow{.34} F[-F]F \end{aligned}$$

2.1.3 Šum

Šum je náhodný, většinou nechtěný signál. Snižuje kvalitu vysílaných dat. Vyskytuje se u analogových i digitálních systémů. Může být jakéhokoliv charakteru. V elektronice je to nechtěná elektromagnetická energie, u fotografií se jedná o obrazový šum, u zvuku se jedná o zvukový šum a na hladině vody díky různému působení sil vznikají vlny. Tyto příklady ukazují nežádoucí varianty.

Jsou však případy, kdy je šum použit k praktické aplikaci. Využívá se jako zdroj náhodných čísel například v kryptografii. Další využití šumu je při procedurálním generování textur. Pokud by byly bez prvků náhody, vše by bylo moc ostré a uměle vypadající. Textury se stochastickým prvkem vypadají přirozeněji [5].

Jak již bylo zníněno, šum je nežádoucí signál, proto je snahou ho eliminovat. Následně když je pro nějaký případ potřeba, je těžké ho získat. Pro generování náhodného šumu je

[2]Převzato z [4, str. 28]

nutné mít specializovaný hardware, který taková čísla dokáže generovat[3]. Tento hardware však není k dispozici v běžném počítači, proto se nahrazuje generátorem pseudonáhodných čísel.

Perlinova šumová funkce

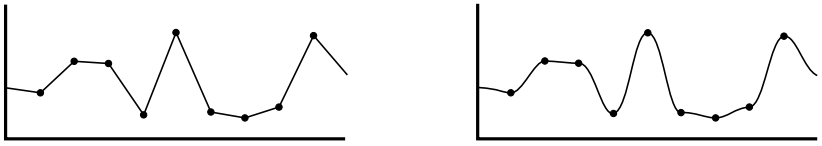
Šumová funkce by měla ideálně generovat bílý šum. Takový šum má frekvenčně neomezené spektrum, takže obsahuje detaily na jakékoliv frekvenční hladině. Proto může být libovolně zvětšován nebo zmenšován, ale jeho výpočet je výkonostně náročný [7]. V mnohých případech není zapotřebí mít detailní šum, například pokud se díváme z dálky, detaily mohou způsobovat alias, což je nežádoucí. Proto v roce 1983 Ken Perlin navrhl metodu generování šumu, která se po něm nazývá, Perlinův šumová funkce. V roce 1997 za ni obdržel cenu Academy Award for Technical Achievement.

Funkce se statisticky nemění vzhledem k otáčení a posunutí, je tedy možné jí začít mapovat jakkoliv natočenou a posunutou. Výsledná textura bude statisticky vždy soběpodobná. Vygenerovaný šum tedy není čistě náhodný, ale koherentní. Lze s ním popsat i fraktální vlastnosti přírody. Dále je funkce opakovatelná, spojitá a má omezené frekvenční spektrum. Tyto vlastnosti se dají využít pro generování textur, kde je žádoucí, aby se neměnila s každým překreslením. Pokud požadujeme dynamické textury, je vhodné, že je funkce spojitá a se změnou v čase nebudou vznikat skoky.

Opakovatelnost zajišťuje pseudonáhodný generátor hodnot. Pseudonáhodný proto, protože pro různé vstupy se zdá, že generuje náhodné hodnoty. Pro stejné vstupy však generuje stejné výstupy. Výstupy jsou v intervalu $\langle -1, 1 \rangle$. Změna vstupu se přímou úměrou projeví i na výstupu. Například pokud vstupní parametry vynásobíme dvěma, výstup bude mít dvojnásobnou frekvenci. Toho se využívá pro skládání šumových funkcí, viz kapitola 2.1.4.

Perlinova funkce je spojitá, ale vypočítává se v diskrétní mřížce. Je tedy nutné interpolovat jednotlivé body nějakou funkcí. Je celá řada způsobů, jak interpolovat vygenerované hodnoty. Nejpoužívanějšími jsou lineární a kosinová interpolace. U většiny interpolačních funkcí jsou tři vstupní parametry, a a b jsou hodnoty, mezi kterými se interpoluje, a x určuje interpolaci a náleží do intervalu $\langle 0, 1 \rangle$ [2].

Lineární interpolace je výkonově nenáročná, ve výsledku však mohou vznikat ostré přechody, což je vidět na obrázku 2.9 vlevo. Lze počítat jednoduše podle rovnice 2.13.



Obrázek 2.9: Lineární interpolace (vlevo), Kosinová interpolace (vpravo), převzato z [2]

$$interpolace = a \cdot (1 - x) + b \cdot x \tag{2.13}$$

U kosinové interpolace se ostré přechody eliminují za cenu mírného snížení rychlosti. Princip zůstává stejný jako u lineární interpolace. Výpočet je možné implementovat podle rovnic 2.14 a 2.15. Výsledná funkce po interpolaci je znázorněna na obrázku 2.9 vpravo.

[3]Někdy je možné využít vlastností hardwaru běžného počítače pro vygenerování náhodných čísel. Lze použít časování diskových operací, čas příchodu paketu ze sítě, poloha kurzoru myši a další.

$$f = \frac{1 - \cos(x - \pi)}{2} \tag{2.14}$$

$$interpolace = a \cdot (1 - f) + b \cdot f \tag{2.15}$$

Perlinův šum dává dostatek volnosti k vytvoření šumové funkce na míru. Může být definovaný v jakékoliv dimenzi, generátor pseudonáhodných čísel může být jakýkoliv a interpolovat můžeme jakoukoliv funkcí. Často se využívá součtu několika Perlinových funkcí a výsledek tohoto součtu bývá označen chybně jako Perlinova funkce.

2.1.4 Skládání šumových funkcí

Pro skládání šumových funkcí se většinou bere jako základ Perlinova šumová funkce. Skládáním funkcí se rozumí sčítání. Sčítají se šumové funkce se změněnou amplitudou a frekvencí. Je to možné zapsat vztahem 2.16. Jednotlivé funkce se často označují jako oktávy.

$$snoise(x, y, f, p, a, n) = \sum_{i=0}^{n-1} a_i \cdot noise(f_i \cdot x, f_i \cdot y) \tag{2.16}$$

$$f_i = 2 \cdot f_{i-1} \tag{2.17}$$

$$a_i = p^i \tag{2.18}$$

- Funkce $snoise(x, y, f, p, a, n)$ představuje součet všech oktáv v bodě $[x, y]$.
- Funkce $noise(f_i \cdot x, f_i \cdot y)$ reprezentuje šumovou funkci (oktávu) v bodě $[x, y]$.
- Parametry x a y určují polohu bodu, ve kterém se počítá šum.
- Parametr f určuje počáteční frekvenci f_0. Pro další oktávy se frekvence přepočítá podle vztahu 2.17.
- Parametr a určuje počáteční amplitudu.
- Parametr p definuje perzistenci. Ta určuje pokles amlitudy každé další oktávy.
- a_i určuje amplitudu oktávy, vypočítá se podle vztahu 2.18.
- Parametr n určuje počet oktáv, které se použijí pro složení výsledku.

Kapitola 3

Návrh

Aplikace bude sloužit pro generování 3D modelů rostlin. Bude vytvořena pod operačním systémem Linux v programovacím jazyce C++. Vygenerované modely budou zobrazeny pomocí grafické knihovny OpenGL do scény, ve které lze tyto modely prohlížet z různých úhlů a vzdáleností. Aplikace by měla být přenosná mezi různými verzemi a distribucemi Linuxu.

Aplikace bude generovat modely rostlin. Rostliny jsou většinou rozvětvené a mají na sobě listy nebo jehličí. Některé rostliny mají viditelné zúžení kmene nebo stvolu, ale jiné se téměř neztenčují. Dále jsou si stejné druhy rostlin podobné, ale nejsou shodné. Podobnost platí i ve výřezech rostlin. Listy jsou podobně veliké na celé rostlině, nezávisí to tedy na stáří větve.

Aplikace by tedy měla zvládat vytvořit rozvětvenou rostlinu, u které se může zužovat hlavní struktura. Dále by měla být možnost vygenerovat listy. Struktura modelu by měla mít prvky náhodnosti, aby výsledek vypadal přirozeněji.

Po zjištění různých metod pro generování bylo vybráno procedurální generování pomocí L-systémů. Zdá se to jako nejlepší volba, protože L-systémy vznikly přímo pro simulaci růstu rostlin. Jsou také jednoduše rozšiřitelné o další funkce, proto bude možné vyvíjet aplikaci postupně. Základem bude deterministický závorkový L-systém. Tento systém byl vybrán jako základ, protože podporuje větvení. Dále se může aplikace rozšířit o generování listů a následně na stochastický L-systém.

Aplikace bude číst L-systém a jeho konfiguraci ze souboru. Popis definice je v podkapitole 3.1. Soubor bude zpracováván lexikální analýzou. Analýza se bude řídit podle gramatiky definované v tabulce 3.1. Dále bude provedena syntaktická analýza podle navržené gramatiky v tabulce 3.2. Výstupem budou data pro generátor řetězce. Vygenerovaný řetězec se následně interpretuje do grafické podoby.

3.1 Gramatika

Soubor obsahující definici L-systému musí obsahovat počáteční axiom a přepisovací pravidla, viz podkapitola 2.1.2. Abeceda systému bude dána implementací želvy. V souboru bude také vhodné provádět nastavení výchozích hodnot želvy.

Základem je možnost ovlivnění úhlu natočení a počet iterací. Bez možnosti ovlivnění těchto parametrů by aplikace nesplňovala účel. Dále by bylo bylo možné ovlivnit ztenčování větví a tloušťku první větve. Gramatika souboru je navržena v tabulce 3.2.

iteration	Konstanta učující počet iterací, které se mají provést.
angle	Konstanta určující defaultní úhel, o který se bude želva otáčet.
reduction	Konstanta určující jak se zmenší v dalším kroku segment. Konstanta *1* znamená žádné zmenšení, tedy následující segment bude mít 100% tloušťky předchozího segmentu a konstanta *0* naopak znamená, že bude mít 0% tloušťky předchozího.
radius	Konstanta určující počáteční tloušťku kmene.
thickness	Konstanta určující tloušťku prvního segmentu. Je vztažena vzhledem k délce segmentu. Například 2 vygeneruje první segment s průměrem rovným dvojnásobku výšky.
axiom	Počáteční řetězec L-systému, který se bude dále rozgenerovávat.
rule	Pravidlo, jak nahradit slovo L-systému.
F	Příkaz želvy určující pohyb vpřed s kreslením čáry.
A–E ∩ G–Z	Slova L-systému, která nemají vliv na želvu.
+	Příkaz pro želvu, aby se otočila vlevo o úhel α okolo vektoru $\vec{U}$.
-	Příkaz pro želvu, aby se otočila vpravo o úhel α okolo vektoru $\vec{U}$.
&	Příkaz pro želvu, aby se naklonila dolů o úhel α okolo vektoru $\vec{L}$.
∧	Příkaz pro želvu, aby se naklonila nahoru o úhel α okolo vektoru $\vec{L}$.
/	Příkaz pro želvu, aby se převalila vlevo o úhel α okolo vektoru $\vec{H}$.
\	Příkaz pro želvu, aby se převalila vpravo o úhel α okolo vektoru $\vec{H}$.
|	Příkaz pro želvu, aby se otočila o 180° okolo vektoru $\vec{U}$.
[	Uložení stavu želvy na zásobník.
]	Načtení stavu želvy ze zásobníku.

Tabulka 3.1: Význam lexémů vstupního souboru

3.2 Generátor řetězce

Při rozgenerovávání L-systému vzniká problém s pamětí. Pro každý krok se vygeneruje nový řetězec, který je větší o aplikovaná pravidla. Dále se v nově vygenerovaném řetězci vygenerují slova, která nejsou v seznamu pravidel a v další iteraci bude potřeba prohledat seznam znovu.

Pro vyhnutí se těmto problémům byla navržena optimalizace, která mění iterativní algoritmus na rekurzivní. Je to možné, protože L-systémy se rozgenerovávají paralelně, proto nezáleží na tom, zda je k dispozici celý řetězec nebo jenom jeho část.

Optimalizace funguje následujícím způsobem; generátor vezme první slovo z axiomu a vyhledá, zda pro něj existuje přepisovací pravidlo. Když neexistuje, je slovo přidáno do výsledného řetězce a pokračuje se dalším slovem. Pokud však existuje, provede se stejný postup nad řetězcem získaným z pravidla jako nad axiomem. To se opakuje, dokud se nedojde do hloubky určené počtem iterací. V této hloubce se přidá řetězec pravidla k výsledku a už se do větší hloubky neprohledávají pravidla a pokračuje se s dalším slovem z předposledního řetězce, viz algoritmus 1.

V_{non} = {START, ITER, ANGLE, REDUCE, AXIOM, RULE, RULES, COMMENT}
V_t = {;, ->, int, float, lsys, string, id}
P = {
START → COMMENT ITER ANGLE REDUCE RADIUS AXIOM RULES
ITER → iteration: int COMMENT
ANGLE → angle: int COMMENT
ANGLE → angle: float COMMENT
REDUCE → reduction: int COMMENT
REDUCE → reduction: float COMMENT
REDUCE → ε
RADIUS → radius: int COMMENT
RADIUS → radius: float COMMENT
RADIUS → ε
AXIOM → axiom: LWORD LWORDS COMMENT
RULES → RULE
RULES → ε
RULE → rule: LWORD -> LWORD LWORDS COMMENT RULES
LWORDS → LWORD LWORDS
LWORDS → ε
LWORD → lsys
COMMENT → ; string \n
COMMENT → ε
}

Tabulka 3.2: V_{non} je abeceda neterminálů a V_t je abeceda terminálů. Terminál *int* reprezentuje celé číslo, *float* číslo s plovoucí desetinou tečkou, *lsys* znak L-systému, *string* textový řetězec, který neobsahuje konec řádku a *id* textový řetězec reprezentující název proměnné. Vysvětlení jednotlivých lexémů je v tabulce 3.1.

3.3 Interpret řetězce

Vygenerovaný řetězec je potřeba převést do geometrické podoby, kterou je potom možné zobrazit. Převod bude probíhat pomocí želví grafiky na transformační matice podle příkazů uvedených v tabulkách 2.1 a 2.2. Matice se potom aplikují na předem vytvořený objekt. Tímto objektem může být cokoliv, ale kmen, větve, stvol a další části rostlin mají většinou přibližně kruhový průřez a jak rostlina roste, tak u vysokých rostlin (například strom nebo keř) je znát i zúžení. U nízkých rostlin (například obilí) není tolik znát zúžení. Pro obilí může být segmentem válec, ale pro strom to nestačí. Nejvhodnějším objektem tedy bude komolý kužel, který bude ovlivnitelný parametrem pro zúžení. Potom bude možné vytvořit více věrnějších modelů rostlin. Dalším parametrem pro věrnější podání modelu je počáteční poloměr segmentu. Stromy jsou širší než květiny, proto bude vhodné mít možnost ovlivnění tohoto parametru.

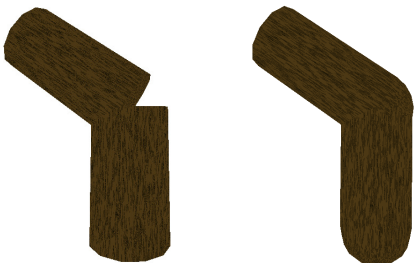
Jak již bylo zmíněno, jako vhodný tvar se zdá být kužel. Může však vznikat problém při natočení. Pokud bude další segment nějakým směrem natočen, docházelo by k narušení celistvosti výsledného modelu vzniklou mezerou, jak je vidět na obrázku 3.1 vlevo. Tomu se lze vyhnout, pokud místo horní podstavy umístíme polokouli. Polokoule bude mít stejný průměr jako horní podstava. Další část bude mít stejný průměr dolní podstavy a při natočení

Algoritmus 1: Optimalizovaný generator

Input: řetězec, iterace
Output: výstup
Data: početIterací, pravidlo[]

```
1  if iterace == početIterací then
2      return řetězec
3  end if
4  foreach řetězec as slovo do
5      if existuje pravidlo[slovo] then
6          výstup += Zavolej se rekurzivně s parametry: (pravidlo[slovo], iterace + 1)
7      else
8          výstup += slovo
9      end if
10 end foreach
11 return výstup;
```

bude hrana dolní podstavy trajektorií kopírovat polokouli z předchozího segmentu, viz obrázek 3.1.



Obrázek 3.1: Vzniklá mezera při natočení (vlevo), zaplnění mezery polokoulí(vpravo)

Pro ověření, zda interpret funguje správně, bude vykreslena trojrozměrná Hilbertova křivka. Pokud bude interpret správný, vykreslená křivka se nikdy neprotne a bude vyplňovat postupně prostor. Ukázka Hilbertovy křivky je na obrázku 2.7.

3.4 Textury

Textura kůry na stromě si je podobná, ale rozpraskání se neopakuje. Rozpraskání vzniká postupným růstem dřeva uvnitř, proto jsou praskliny většinou podél kmene a větví.

Jelikož se vzled rozpraskání kůry neopakuje, bude vhodné použít nějakou procedurální metodu. Kůra by se dala považovat za fraktál. Perlinův šum lze použít na vytvoření fraktálních útvarů, proto bude vhodný i zde. Bylo by možné vytvořit texturu pomocí dvourozměrného Perlinova šumu, ale pro zjednodušení mapování textury na objekt se využije trojrozměrná funkce.

Rozpraskání kůry ve směru větví lze vytvořit roztažením šumu na lokální ypsilonové ose segmentu. Textura se však bude opakovat na každém segmentu, což je nežádoucí. Další možností je generovat texturu v globální scéně, ale praskliny budou ve směru globální osy ypsilon bez ohledu na natočení segmentu. Bude tedy potřeba vygenerovat proužky na segmentu a následně na ně použít Perlinův šum počítaný podle globálních souřadnic. Textura by se potom mohla podobat kůře stromu.

Kapitola 4

Implementace

Aplikace je vytvořena v programovacím jazyce C++, využívá normu c++11. Využité grafické knihovny jsou OpenGL, GLEW, GLFW a GPUEngine, dále je využita ještě matematická knihovna GLM, která umožňuje používat matice a vektory jako v shaderech GLSL. Program byl vyvíjen pod operačním systémem GNU/Linux, distribuce Archlinux.

Program je rozdělen na několik částí, kterými jsou lexikální analyzátor implementovaný třídou *Lexical*, syntaktický analyzátor implementovaný třídou *Syntax*, generátor řetězce implementovaný třídou *Gen*, interpret řetězce implementovaný třídou *GenPoint*, reprezentace želvy implementovanou třídou *Turtle* a L-systém implementovaný třídou *Info*. Pro lepší názornost byl vytvořen graf, který je na obrázku 4.1.

4.1 Lexikální analyzátor

Lexikální analyzátor je implementován ve třídě *Lexem*. Pracuje jako konečný automat, který čte vstupní soubor znak po znaku a rozeznává lexémy. Těmi jsou klíčová slova, identifikátory, čísla a znaky L-systému. Lexémy jsou následně převedeny na tokeny, které se předávají syntaktickému analyzátoru.

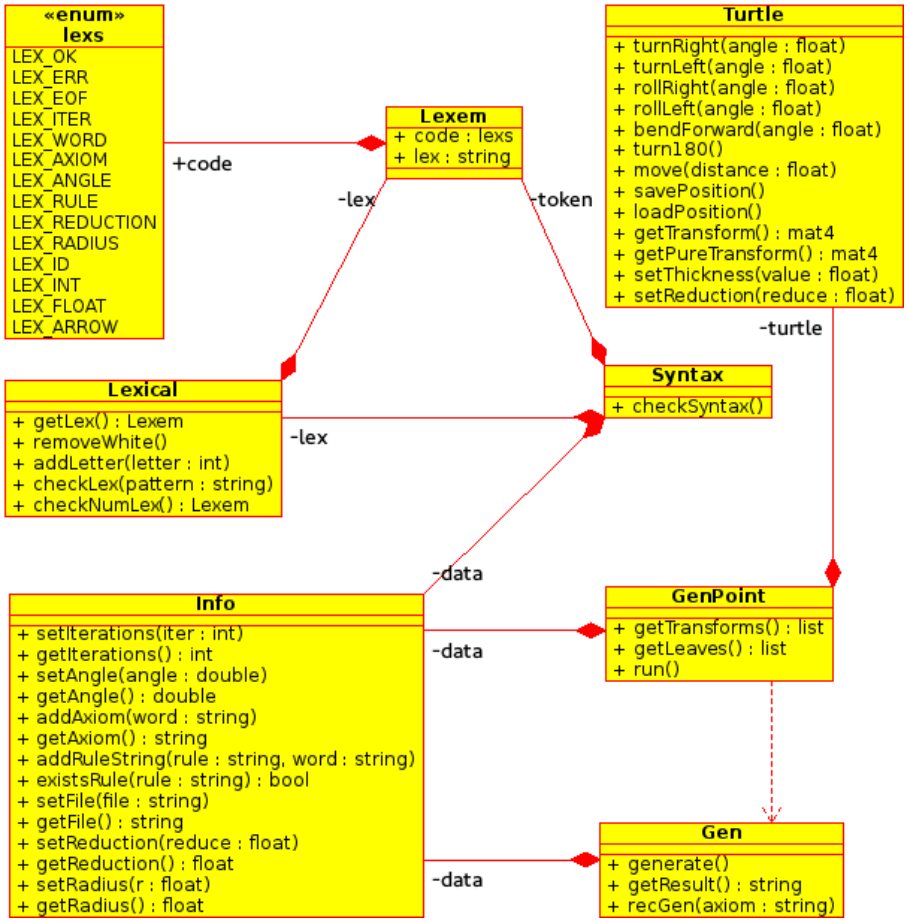
4.2 Syntaktický analyzátor

Úkolem syntaktického analyzátoru je kontrola správnosti syntaxe vstupního souboru. K tomu využívá metodu rekurzivního sestupu. To znamená, že jednotlivé neterminály z množiny V_{non} (definované v gramatice 3.2) jsou implementovány v metodě a volají metodu *getLex* z třídy *Lexem*. Následně se zkontroluje, zda je vrácený token očekávaného typu. Během kontroly se inicializují struktury uschovávající informace o L-systému, kterými jsou axiom, přepisovací pravidla, počet iterací a další.

4.3 L-systém

Třída *Info*, implementující L-systém, je důležitá část aplikace. V syntaktickém analyzátoru se do ní uloží načtené hodnoty ze vstupního souboru. Dále v generátoru řetězce se z ní načítají pravidla a v interpretu se získá defaultní úhel natáčení.

Výběr pravidla se provádí na základě jejich existence. Pokud žádné pravidlo neexistuje, jako následovník se použije předchůdce. Pokud existuje alespoň jednoho pravidlo pro aktuální slovo, nerozlišuje se, zda se jedná o deterministický nebo stochastický L-systém.



Obrázek 4.1: Graf implementace aplikace

Pravidlo se vybere vygenerováním náhodného čísla, které určí, kolikáté pravidlo z vybraných se použije. Pokud existuje právě jedno pravidlo pro aktuální slovo, vždy se vybere. Takový výběr pravidla se tedy bude chovat deterministicky.

Pro vygenerování čísla se používá defaultní funkce *rand*. Počáteční hodnota (seed) je nastavena na aktuální čas v sekundách (UNIX timestamp). Tím se při každém spuštění zajistí jinak vygenerovaný model. Pro získání stejného modelu je možné ovlivnit počáteční hodnotu parametrem *-r* a číselnou hodnotou.

4.4 Generátor řetězce

Po provedení lexikálního a syntaktického analyzátoru je možné začít zpracovávat načtená data. Vezme se axiom a začne se od začátku znak po znaku procházet. Pro každé slovo L-systému se projde seznam přepisovacích pravidel. Pokud existuje alespoň jedno přepisovací pravidlo pro aktuální slovo, aplikuje se pravidlo získané z L-systému přepsáním původního slova. Pokud pravidlo neexistuje přidá se znak přímo k výslednému řetězci. Takto se bude procházet až do hloubky určené počtem iterací. Pro implementaci byla využita navržená optimalizace z kapitoly 3.2.

4.5 Interpret řetězce

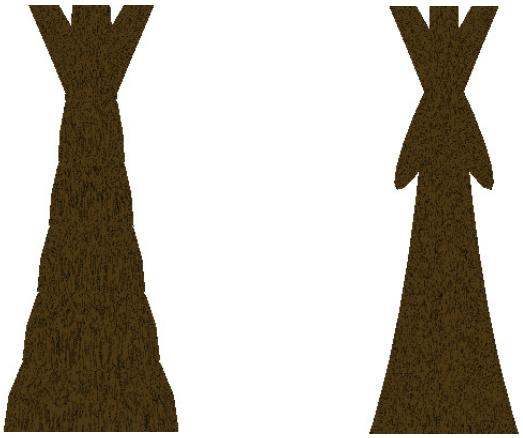
Řetězec je do grafické podoby interpretován želví grafikou. Ta vygeneruje transformační matice, kterými se natáčí kužel. Celá rostlina je reprezentovaná jedním kuželem, který se při vykreslování natáčí pomocí vygenerovaných transformačních matic. Tento kužel se generuje pomocí funkce *createCone*. Je možnost v programu místo této funkce zavolat *createCube*, která místo kužele vygeneruje krychli. To je zde připraveno pro pozdější rozšíření programu.

Kužel se vytvoří podle parametrů zaslaných funkci *createCone*. Těmito parametry jsou počet stran, které budou po obvodu kužele, poloměr, který určuje tloušťku kužele, poměr, jak se zmenší horní podstava a seznam, do kterého se vygenerují body. Místo spodní podstavy je vygenerovaná polokoule, aby nevznikaly mezery při natočení. Porovnání tvarů segmentu je na obrázku 4.2. Spodní podstava byla vybrána, protože když polokoule byla místo horní podstavy, nastal problém při velkém koeficientu zúžení segmentu. Polokoule byla občas větší než další kužel a vytvořila na kmeni nežádoucí zvlnění, viz obrázek 4.3 vlevo. Pokud polokoule byla použita místo spodní podstavy, nastal také problém. Kužel se zmenšuje pouze po stranách, ale výška zůstává stejná. Po určitém zúžení se v rozvětvení začnou objevovat nežádoucí artefakty, viz obrázek 4.3 vpravo. Tyto artefakty se objevují i tehdy, pokud je polokoule místo horní podstavy, jsou pouze ve směru větve. Lze si je snadno splést s vygenerovanými větvemi. Tomuto problému by se dalo vyhnout vytvořením objektů pouze pro tyto přechody, které by se zmenšovaly ve všech směrech nebo vygenerovat strom jako celek. Při generování stromu jako celek by bylo možné programově ovlivnit polokouli jinou transformací než zbytek kužele, aby se artefakty nevytvářely.

Obrázek 4.2: Segment se zakulacením nahoře (vlevo), segment se zakulacením dole (vpravo)

Hlavní část interpretace se provádí ve třídě *GenPoint*, kde se rozeznávají jednotlivá slova L-systému jako příkazy pro želvu. Želva je implementovaná ve třídě *Turtle*. Pokud se v interpretovaném řetězci objeví znak *F*, znamená to vykreslení kmene, což je zmiňovaný kužel nebo krychle. Od želvy se získá aktuální transformační matice metodou *getTransform* a uloží se do seznamu s ostatními maticemi pro kmen. Potom je potřeba zajistit posunutí želvy. To se provede zavoláním metody *move*, kde se interně provede zúžení a posun o zadanou délku. Správnost interpretace byla ověřena vykreslením Hilbertovy křivky. Výsledek je vidět na obrázku 2.7.

Pro znaky rotace (uvedené v tabulce 2.2) se v metodě *rotate* rozhodne, kterým směrem se má želva otočit a zavolá se její příslušná metoda pro provedení příslušné rotace. Dále bylo implementováno jednoduché generování listů. Znak *L* znamená vykreslení listu rostliny. Na začátku je orientován stejně jako kužel a řídí se stejnou transformační maticí jako kužel, pouze se neaplikuje zmenšení. Vykreslení se provede získáním aktuální transformační matice želvy bez jejího přesunu a uložením matice do seznamu s ostatními listy.



Obrázek 4.3: Nechtěné zvlnění povrchu (vlevo), vzniklé artefakty při rozvětvení (vpravo)

4.6 Želví grafika

Želva je implementována ve třídě *Turtle*. Jsou zde implementované metody pro interpretaci všech symbolů, které ovlivňují pohyb a pozici želvy. Rozhodnutí, o který symbol se jedná, se provádí již ve třídě *GenPoint* a potom se zavolá jenom přislušná metoda provádějící akci. U většiny metod provádějících akce je požadován parametr, kterým se ovlivňuje velikost úhlu, o který se má želva natočit, nebo vzdálenost, o kterou se má želva posunout. Pro získání transformační matice jsou implementovány dvě metody. První metodou je *getTransform*, která vrací matici ovlivněnou postupným zužováním. Využívá se při vytváření kmene a větví. Druhou metodou je *getPureTransform*, která vrací matici bez aplikovaného zúžení. Ta se využívá při vytváření listů.

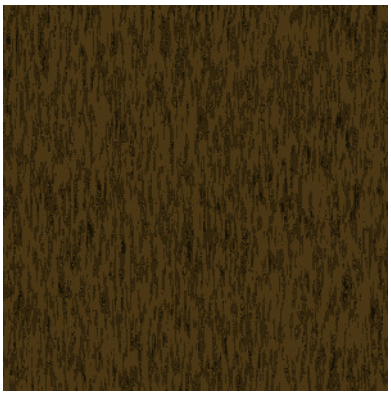
4.7 Textury

Textury jsou generované procedurálně pomocí Perlinovy šumové funkce. Tato funkce je implementována pro provádění v grafickém shaderu pomocí jazyka GLSL.

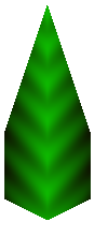
Bylo vypozorováno, že stromy mají většinou popraskanou kůru podélně, nezávisle na směru růstu větve. Proto se nejprve vygenerují proužky pomocí dvourozměrného Perlinova šumu. Je využita funkce pro generování trojrozměrného šumu, ale ypsilonová složka je vždy vynulovaná. Tím vzniknou rovné proužky podél větví, které jsou použity pro ovlivnění trojrozměrného šumu, který je vygenerován následně. Tím vzniknou podélné praskliny. Výsledná textura je zobrazena na obrázku 4.4.

Textura listu byla vytvořena jednoduše bez prvků náhody. Nejsvětlejší je v místech, kde bývá „hlavní žíla“ listu. Rozvětvení do menších žilek je provedeno pomocí funkce sinus. Mírné ztmavení na okrajích vytváří iluzi prohnutí listu, při pohledu ze strany však list zmizí, protože je plochý. Výsledný list s texturou je zobrazen na obrázku 4.5

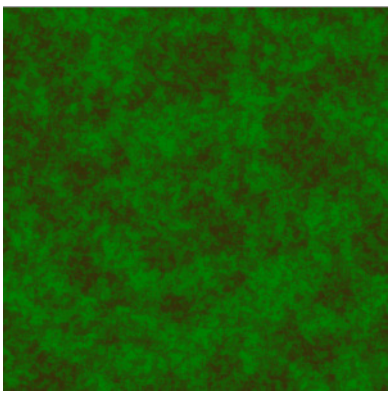
Textura trávy na zemi je vygenerovaná pomocí dvou dvourozměrných Perlinových šumů. Implementace šumu ořezává záporná čísla na nulu, proto se pozice předává v absolutní hodnotě. Tim vznikl v bodě spojení kříž, kde byla textura pouze převrácená. Proto se k šumu přičítá ještě druhý šum o něco posunutý, tím vznikne náhodně vypadající textura, která se neopakuje. Vygenerovaná textura je zobrazena na obrázku 4.6.



Obrázek 4.4: Textura kůry vygenerovaná Perlinovou šumovou funkcí



Obrázek 4.5: List vygenerovaný aplikací



Obrázek 4.6: Textura trávy vygenerovaná Perlinovou funkcí

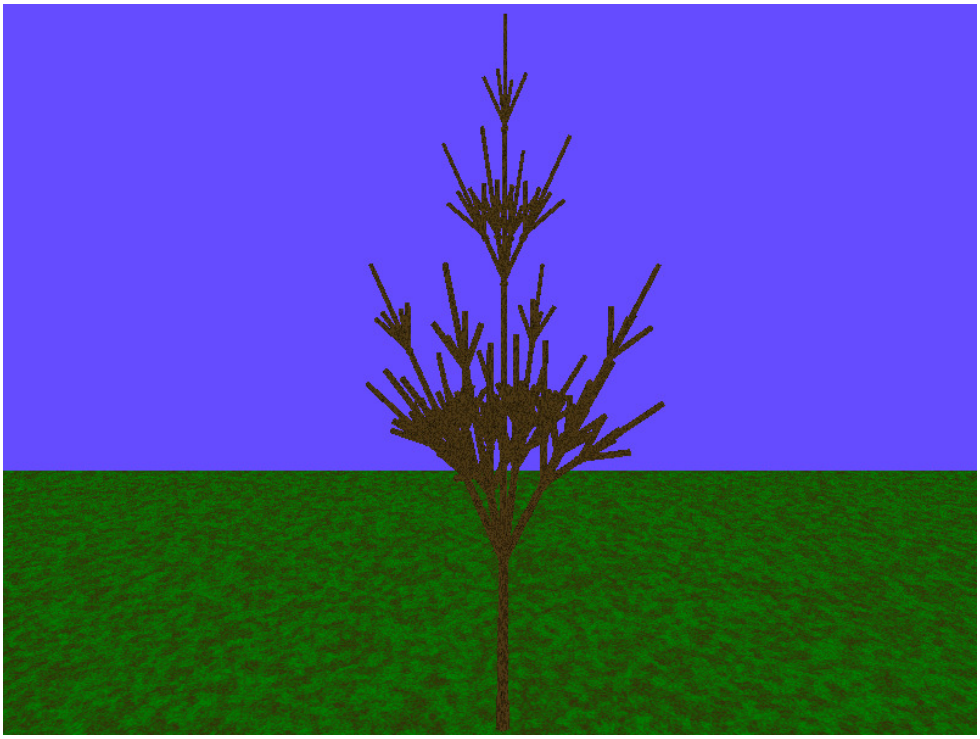
Kapitola 5

Dosažené výsledky

V této kapitole budou ukázány výstupy aplikace a budou porovnány jednotlivé metody generování.

Bylo navrhováno zkontrolovat správnost implementace interpretu pomocí Hilbertovy křivky. Kontrola proběhla úspěšně a výstup je zobrazen na obrázku 2.7. Na tomto obrázku je však výstup pozměněn. Je odstraněná plocha reprezentující zem, pozadí je nastavené na bílou barvu a křivka je obarvena jednoduchou texturou, kde se souřadnice vykreslovaného bodu reperzentují jako barva. Tímto testem se však netestovaly textury, ale správná rotace a posun želvy – správné natočení a umístění válců.

Základem je interpret závorkového L-systému, který pracuje v trojrozměrném prostoru. Na obrázku 5.1 je trojrozměrná interpretace závorkového L-systému. Jedná se o trojrozměrnou variantu L-systému vyobrazeného na obrázku 2.8.



Obrázek 5.1: Trojrozměrná reprezentace L-systému z obrázku 2.8

5.1 Porovnání metod

Na obrázku 5.2 je vytvořen model rostliny podobný keři. Je vygenerován závorkovým L-systémem bez jakéhokoliv rozšíření. V horní části modelu je vidět, že detaily již nejsou rozeznatelné. Model je generovaný podle L-systému definovaném vstupním souborem 5.1.

Obrázek 5.2: Struktura podobná keři vygenerovaná závorkovým L-systémem bez rozšíření

Po rozšíření L-systému o zužování kmene jsou detaily zřetelnější při stejném L-systému, viz obrázek 5.3. Model je vygenerovaný podle stejného L-systému jako předchozí příklad. Takový model by již šel využít v zimní krajině, je však moc symetrický. Pokud by jich bylo více vedle sebe, bylo by to nápadné a celá scéna by tak vypadala uměle.

Pro ještě lepší výsledek byly implementovány listy, viz obrázek 5.4. Model byl vytvořen podle L-systému 5.2. Nyní je možné využít modelovaný keř v různých scénách, ale jako u obrázku 5.3 zde přetrvává problém symetrie.

Pro odstranění symetrie z modelů byly implementovány stochastické L-systémy, viz obrázek 5.5. Model byl vytvořen podle L-systému 5.3. Pro porovnání: model na obrázku 5.6 je vygenerovaný podle stejného vstupního souboru, model však vypadá odlišně. Keř se tedy nemusí vygenerovat pokaždé stejně. Při použití pseudonáhodného generátoru se stejně nastaveným počátkem však dostaneme vždy stejný model. Toho lze využít například ve hrách. Pokud by hra při každém načtení vygenerovala jinak vypadající modely, hráč by se neměl podle čeho orientovat a tím by hra ztrácela na hratelnosti (pokud by to nebyl záměr).

Další příklady rostlin jsou na přiloženém CD.

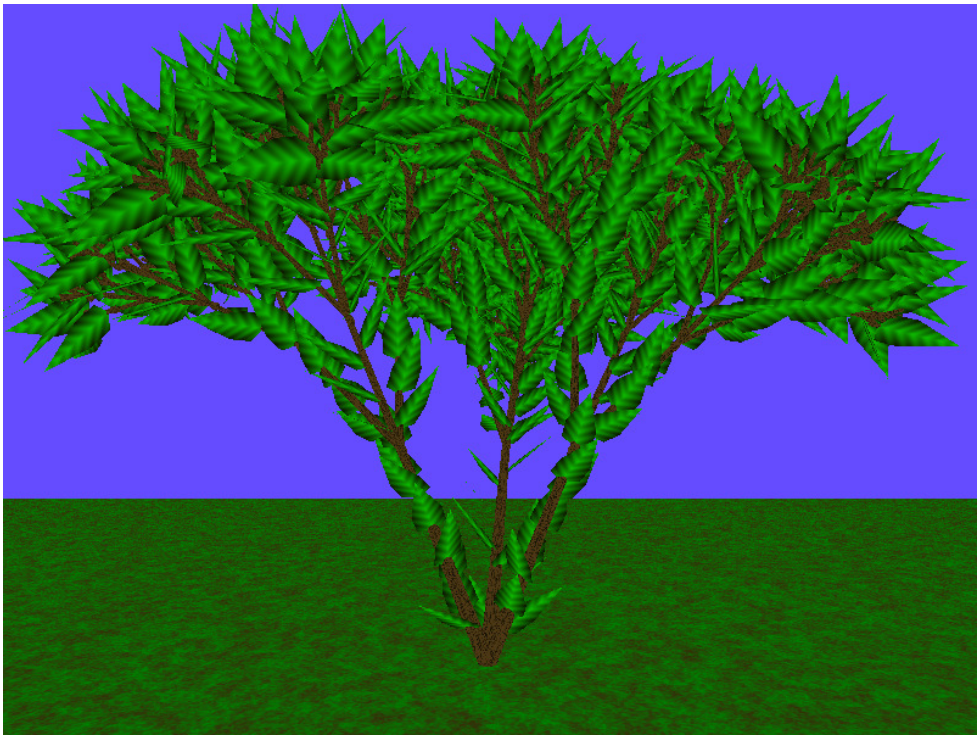
```
1  ;number of iterations
2  iteration: 7
3
4  ;default angle of rotation
5  angle: 22.5
6
7  ;axiom
8  axiom: A
9
10 ;production rules
11 rule: A -> [\&FA]/////[\&FA]///////[\&FA]
12 rule: F -> S ///// F
13 rule: S -> F\end{verbatim}
```

Tabulka 5.1: Zdrojový kód L-systému struktury bez listů

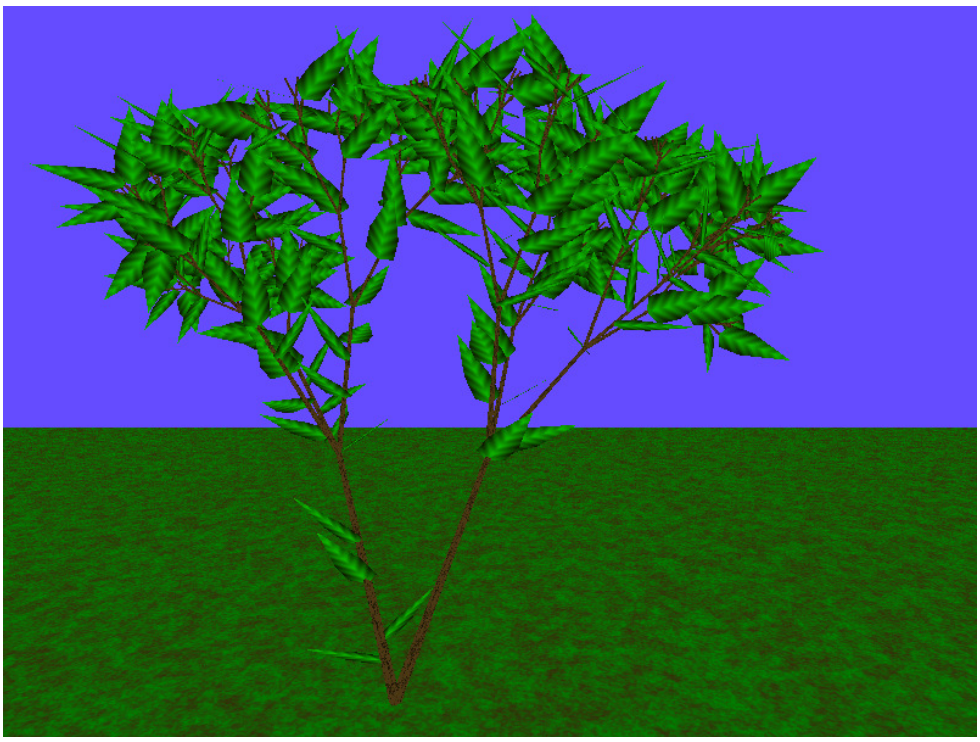
```
1  ;počet iterací
2  iteration: 7
3
4  ;default angle of rotation
5  angle: 22.5
6
7  ;next piece's thickness will be 95% of current thickness
8  reduction: 0.95
9
10 ;initial radius
11 radius: 0.3
12
13 ;axiom
14 axiom: A
15
16 ;production rules
17 rule: A -> [&FXA]/////[&FXA]///////[&FXA]
18 rule: F -> S ///// F
19 rule: S -> FX
20 rule: X -> [^^L]
```

Tabulka 5.2: Zdrojový kód L-systému popisující keř s listy

Obrázek 5.3: Struktura podobná keři se ztenčováním větví



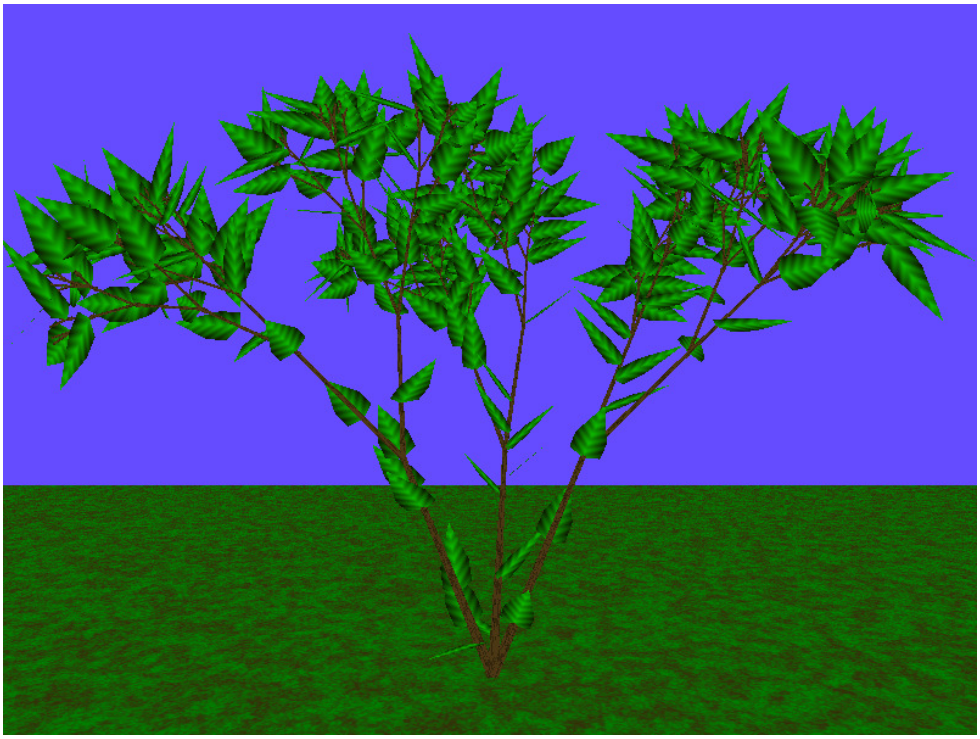
Obrázek 5.4: Struktura podobná keři s listy



Obrázek 5.5: Struktura podobná keři s náhodným větvením

```
1  ;number of iterations
2  iteration: 7
3
4  ;default angle of rotation
5  angle: 22.5
6
7  ;axiom
8  axiom: A
9
10 ;production rules
11 rule: A -> /////[&FXA]///////[&FXA]
12 rule: A -> [&FXA]////////////[&FXA]
13 rule: A -> [&FXA]/////[&FXA]///////[&FXA]
14 rule: A -> [&FXA]/////[&FXA]///////
15 rule: F -> S ///// F
16 rule: S -> FX
17 rule: S -> F
18 rule: X -> [^^L]
19 rule: X -> [^^^L]
```

Tabulka 5.3: Zdrojový kód L-systému popisující keř se stochastickým větvením



Obrázek 5.6: Struktura podobná keři s náhodným větvením

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit aplikaci, která bude generovat modely rostlin a následně navrhnout rozšíření metody generování. Dále se měly tyto metody porovnat.

Byla vybrána metoda generování rostlin podle L-systémů. Základní deterministický L-systém umí popsat pouze křivku v prostoru. Neumí tedy vytvořit rozvětvenou strukturu. Proto byl implementován generátor rostlin založený na interpretaci závorkových L-systémů, který větvení již podporuje. Pro předání L-systému aplikaci byl navržen vstupní konfigurační soubor, který slouží pro zápis L-systému a jeho nastavení, viz sekce 3.1.

Následně bylo implementováno rozšíření želvy, aby se vykreslovaný kmen zužoval. Potom stromové struktury vypadají reálněji. Pro ještě reálnější modely bylo implementováno rozšíření umožňující zobrazit listy.

Další implementované rozšíření L-systému bylo stochastické chování. V přírodě se běžně neobjevují rostliny, které by byly shodné s jinými nebo se větvily v každém rozvětvení úplně stejně. Pomocí deterministického chování toto nelze ovlivnit. Stochastické L-systémy mohou mít více pravidel pro jedno slovo, proto každé rozvětvení může vypadat jinak. Vygenerované modely tímto způsobem by se již mohly použít pro scénu s realisticky vypadajícími stromy. Není však možné dynamicky ovlivnit tvar listů a barvu textur.

Při použití optimalizace generátoru řetězce navržené v sekci 3.2 se zrychlilo generování řetězce z řádu desítek minut na jednotky sekund pro stejnou konfiguraci. Lze tak vytvářet detailnější modely za nižší čas.

Vytvořená aplikace splnila daný cíl prace.

Možnost jejího dalšího vývoje je implementace parametrických L-systémů, L-systému generujícího listy podle zápisu v souboru a dalších pokročilejších L-systémů. Dále by se mohlo vylepšit zobrazování scény přidáním sky-boxu, shadow mappingu, bump mappingu. Po rozšíření L-systému o možnost generovat listy bude možnost podobným způsobem určit tvar kmene.

Literatura

[1] Eastwood, B. S.; Maxwell, B. A.; Skrien, D. J.; aj.: CS151: Lab 11 [online]. http://cs.colby.edu/courses/S12/cs151-labs/L11-3DTurtle.php, 2012 [cit. 2015-02-11].

[2] Elias, H.: Perlin Noise [online]. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm, [cit. 2015-05-10].

[3] Prusinkiewicz, P.; Hammel, M.; Měch, R.; aj.: The Artificial Life of Plants [online]. http://algorithmicbotany.org/papers/l-sys.sig95.pdf, 1995-08-23 [cit. 2014-09-17].

[4] Prusinkiewicz, P.; Lindenmayer, A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990, ISBN 978-0-387-97297-8.

[5] Tišnovský, P.: Perlinova šumová funkce a její aplikace [online]. http://www.root.cz/clanky/perlinova-sumova-funkce-a-jeji-aplikace, 2007-03-20 [cit. 2015-03-13].

[6] Zelinka, I.; Včelař, F.; Čandík, M.: *Fraktální geometrie, principy a aplikace*. BEN – technická literatura, 2006, ISBN 80-7300-193-4.

[7] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2010, ISBN 80-251-0454-X.