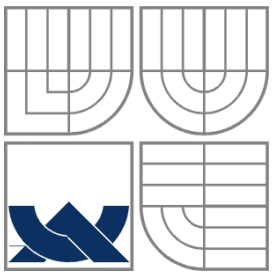
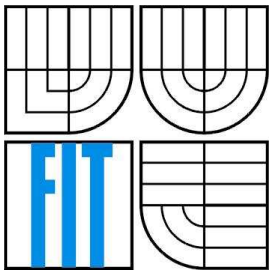


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

L-SYSTÉMY V 3D GRAFIKE

L-SYSTEMS IN 3D GRAPHICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JÁN MARČEK
AUTHOR

VEDOUCÍ PRÁCE — ING. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

Abstrakt

Modelovaním pomocou L- systémov vzniká veľké množstvo problémov, ktoré sú analyzované a riešené v tejto práci. V práci je rozobratá tematika L – systémov, s využitím jME grafického rozhrania, ktorá vedie k vytvoreniu aplikácie pre generovanie krajinky.

Abstract

Modelling by using L-systems opens many kinds of problems, witch are analyzed and discussed in this work. In this work we talk about L – systems using jME graphics interface witch leads to application for generating landscape.

Kľúčové slová

L – systémy, Lindenmayerove systémy, java Monkey Engine, jME, generovanie terénu, reťazec,slovo, krajinka.

Keywords

Lindenmayer systems, java Monkey Engine, L- systems, jME, generating terrain, string, word, landscape.

Citace

Marček Ján: L-systémy v grafike, bakalářská práce, Brno, FIT VUT v Brně, rok 2009

L-systémy v 3D grafike

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Aleša Láníka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Ján Marček
2009

Poďakovanie

Chcel by som poďakovať asistentovi Ing. Alešovi Láníkovi za poskytované cenné rady a za čas, ktorý mi venoval pri riešení danej problematiky. Ďalej by som chcel poďakovať svojej rodine, ktorá pri mne stála.

Obsah

Zoznam obrázkov

1 Úvod

Pre človeka bola vždy veľká výzva vytvoriť systém, ktorý by dokázal modelovať a vykresľovať veľké prostredia, ako sú napríklad mestá. Mestá sú veľmi komplexné a na ich modelovaní sa odzrkadľujú historické, kultúrne, ekonomické a sociálne zmeny, ktoré nastali počas stáročí . L – systémová tematika sa práve v tejto dobe stala aktuálnou pretože, pre ich konštrukciu potrebujeme mať už spomínané informácie. Pre ich zachytenie spotrebovávame veľkú kapacitu pamäte, činnosti procesora a výkon grafickej karty, ktorý sa rozvíja najviac v tejto dobe.

L – systémy majú svoje využitie či už vo vzdelávacích programoch, alebo v programoch s cieľom vytvoriť simuláciu reálnej krajiny, alebo pri návrhu infraštruktúry mesta. Svoje miesto majú aj v zábavnom priemysle pri tvorbe hier, ktoré vyžadujú rýchlu a rôznorodú tvorbu prostredia. Svoju úlohu zohrávajú aj v biologickom výskume pri štúdiu buniek.

Vizualizácia veľkých modelov, komplexných systémov, má dlhú tradíciu v počítačovej grafike. Množstvo týchto modelov sa objavuje aj v prírodnom fenoméne.

Táto práca sa zaoberá vytváraním malej krajinky, v ktorom sa nachádza mesto tvorené jednoduchými objektmi a ktoré je generované za pomoci L – systémov.

V práci tieto L – systémy popíšem z teoretického hľadiska, zároveň poskytnem komplexný pohľad na danú problematiku, čo môže poslúžiť pre rozšírenie vedomostí k ďalšiemu využitiu pri tvorbe grafických aplikácií.

Dôležitým bolo aj zapracovanie rozhrania Java Monekey Engine, ktoré bolo použité pri riešení danej tematiky. V práci sa zaoberám používaním jeho základných funkcií.

V práci teda dochádza k spojeniu týchto dvoch nástrojov tak, aby mohol vzniknúť výsledný model.

Kapitola číslo 3 vysvetľuje postupnú konštrukciu L – systémov, preberá problémy, ktoré sa vyskytli s cieľom ich analýzy a riešenia.

V závere poskytnem výsledné riešenie a načrtnem možnosti pre rozpracovanie problematiky v ďalších prácach.

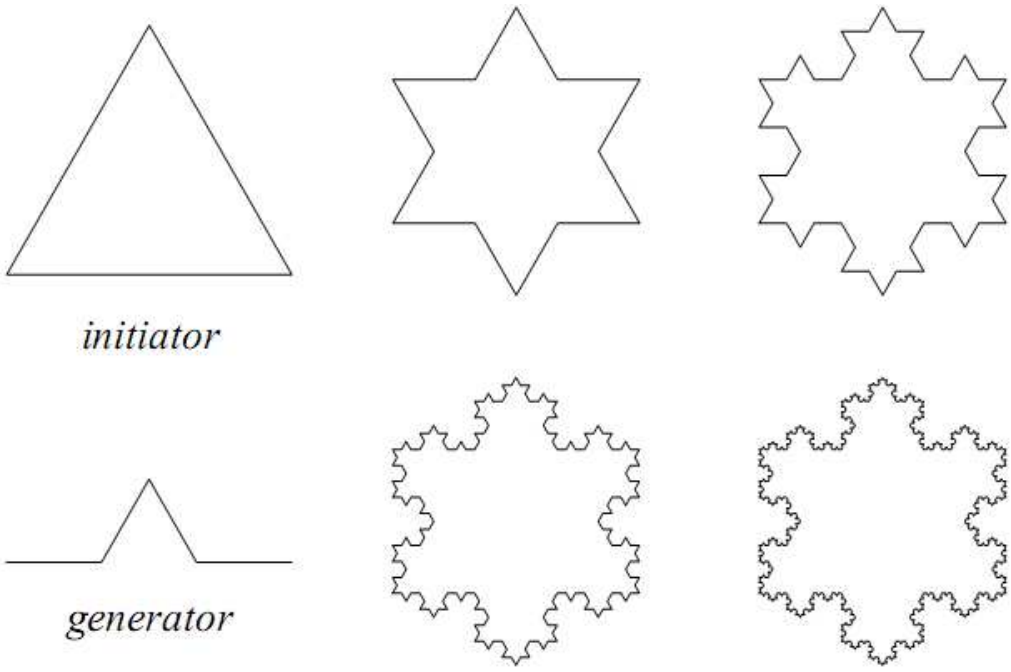
2 Grafické modelovanie pomocou L – systémov

Lindenmayerove systémy, alebo skrátene L – systémy – predstavujú matematickú teóriu pre vyvíjanie rastlín. [4] Pôvodne nezahŕňali toľko detailov ohľadom všeobecného modelovania. Dôraz bol kladený na rastlinnú topológiu, a to hlavne na susedské vzťahy medzi bunkami alebo väčšími rastlinnými modulmi. Ich geometrická stránka bola za rámcom teórie. Následne, niekoľko geometrických interpretácií za pomoci L – systémov bolo navrhnutých s tým, že zmenia ich všestranné použitie ako nástroj pre modelovanie rastlín. V tejto kapitole rozoberiem tzv. korytnačiu problematiku (turtle interpretation) v ktorej som našiel ťažisko pri riešení mojej témy.

2.1 Prepisovacie systémy

Hlavnou ideou L – systémov je prepisovanie. Všeobecne, prepisovanie je technika pre definovanie komplexných objektov postupným nahradzovaním časti z jednoduchého počiatočného objektu použitím množiny *prepisovacích pravidiel*. *Snehová vločka (snowflake curve)* je klasickým príkladom grafického objektu vytvoreného z definovaných výrazov v prepisovacích pravidlách. Je znázornená na obrázku 7. Konštrukcia bola navrhnutá v roku 1905 *Von Kochom* a je nasledujúca :

Začíname s dvomi tvarmi, sú to *inicializátor* a *generátor*. Generátor je orientovaná zalomená čiara vytvorená z N rovnakých strán s dĺžkou r. Každý tento kúsok konštrukcie začína s lámaním čiary a spočíva v nahradzovaní každého rovného intervalu kópiou generátora, zmenšuje a vytláča niektoré koncové body a tým nahradzuje intervaly. (Obrázok 1 - konštrukcia snehovej vločky.)



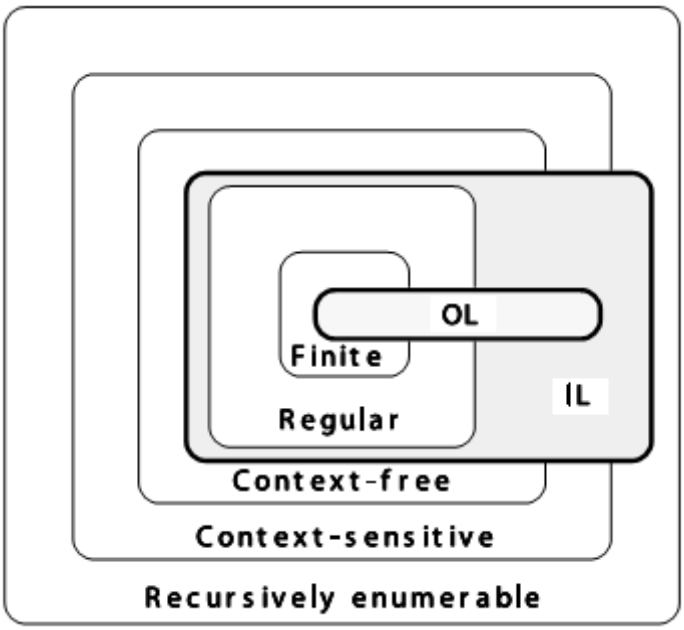
Obrázok 1 - konštrukcia snehovej vločky

[4]

Kým Kochova konštrukcia rekurzívne nahradzuje otvorené polygóny, prepisovací systém ich prevádza na iné objekty, podľa toho akého tvaru boli vytvorené. Napríklad, štúdie Wolfráma generujú vzorky, ktoré sú tvorené prepisovacími elementami obdĺžnikových polí. Podobný prepisovací – polový mechanizmus je základným kameňom pre Conwayovú populárnu hru meno život. [4]. Dôležitou časťou je výskum venovaný rôznym grafovo – prepisovacím systémom.

Značne veľká časť štúdia a najlepšieho pochopenia prepisovacích systémov je v chápaní znakových reťazcov. Prvá formálna definícia, bola podaná začiatkom tohto storočia Thueom, ale široký záujem v prepisovaní reťazcov nastal neskôr v roku 1950 Chromského prácou v oblasti formálnej gramatiky. Aplikoval návrh, ktorého základ spočíva v prepisovaní syntaktických prvkov z prirodzených jazykov. Niekoľko rokov neskôr Backu a Naur uviedli nový základ prepisovaní za pomoci formálnej definície programovacích jazykov (ALGOL-60). Rovnocennosť BNF formy a kontextovo – voľných tried z Chomského gramatiky bola veľmi rýchlo rozpoznaná a bola aplikovaná do počítačovej vedy. Stredobodom pozornosti bola množina reťazcov – nazývané formálne jazyky – a metód pre generovanie, rozpoznávanie a ich transformáciu.

V roku 1968 biológ, Aristid Lindenmayer, uviedol nový typ reťazcovo-prepisovacích mechanizmov, ktoré boli následné pomenované termínom L-systémy [4]. Chomského gramatika a L-systémy sa nachádzajú v metóde pre aplikovanú tvorbu.



Obrázok 2 - Vzťahy medzi Chomského triedami jazykov a jazykov generovanými pomocou L - systémov. Symbol OL a IL označujú jazyky ktoré sú generované kontextovo-voľnými a kontextovo-sensitívnymi jazykmi.

.

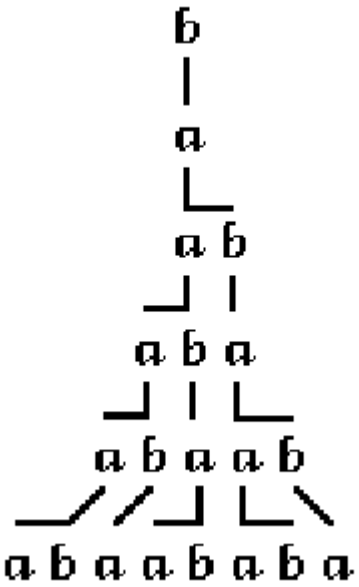
Chomského gramatika je aplikovaná postupne, zatiaľ čo L – systémy sú aplikované paralelne a postupne premiestňujú všetky písmenká v reťazci. Tento rozdiel reflektuje biologickú povahu L – systémov. Produkty sú získané v oblasti buniek v multibunkovom organizme, v ktorom sa vyskytuje väčší počet takýchto oblastí v rovnaký čas. Aplikácia paralelného vytvárania má dôležitý vplyv na formálne vlastnosti prepisovacích pravidiel. Napríklad, jazyky ktoré môžu byť generované bez kontextových L – systémaov (nazývame ich OL - systémy), ale nemôžu byť bez kontextovej Chomského gramatiky (Obrázok 2 - Vzťahy medzi Chomského triedami jazykov a jazykov generovanými pomocou L - systémov. Symbol OL a IL označujú jazyky ktoré sú generované kontextovo-voľnými a kontextovo-sensitívnymi jazykmi..)

2.2 DOL – systémy

Táto časť hovorí o jednoduchej triede L – systémov. Tie ktoré sú deterministické a bez- kontextové voláme DOL – systémy. Vysvetľovanie začnem na príklade v ktorom uvediem hlavnú myšlienku pomocou intuitívnych pojmov.

Uvažujeme o reťazci (slovo) zostavené len z dvoch písmen a to *a* a *b*, ktoré sa môžu vyskytnúť niekoľkkrát v danom reťazci. Každé písmeno je spojené s prepisovacím pravidlom. Pravidlo *a –> ab* hovorí, že znak *a* je nahradený reťazcom *ab*, a pravidlo *b –> a* hovorí že písmeno *b* je nahradzované

písmenom *a*. Prepisovací proces začína reťazcom, ktorý nazývame *axióm*. Predpokladáme, že pozostáva z jedného písmena a to *b*. V prvom kroku odvodzovania (prvý krok prepisovania) axióm *b* je nahradený písmenom *a* za použitia pravidla *b –>a*.



Obrázok 3 - príklad odvodzovania v DOL – systémoch

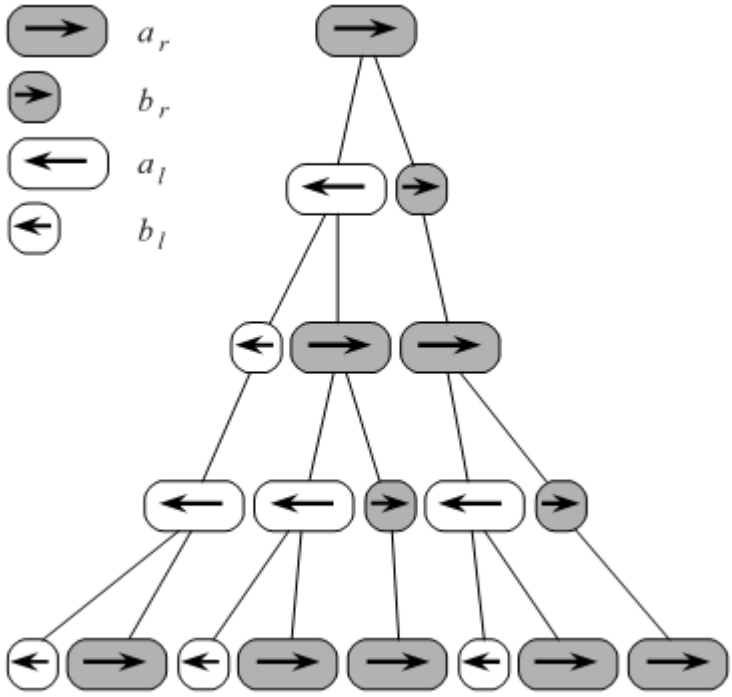
Druhým krokom bude *a* nahradené reťazcom *ab* pomocou pravidla *a –> ab*. Slovo *ab* pozostáva z dvoch písmen, ktoré sa postupne nahradzujú v ďalších odvodeniach. Toto *a* je nahradené *ab*, *b* je nahradené *a*. Výsledkom je *aba* reťazec. Zjednodušene, reťazec *aba* vytvára *abaab*, ktorý vytvára *abaababa* a ten *abaababaabaab* tak ako je znázornené na obrázku (Obrázok 3 - príklad odvodzovania v DOL – systémoch) .

Formálna definícia opisujúca DOL – systémy a ich aplikácia je uvedená nižšie. (pre detaily [3]).

Nech *V* označuje abecedu, V^* označuje množinu všetkých reťazcov nad abecedou *V*, a V^+ je množina všetkých neprázdnych reťazcov nad *V*. Reťazec OL – systém je usporiadaná trojica $G = \langle \mathrm{V}, \omega, P \rangle$ kde *V* je abeceda systému, $\omega \in V^+$ je neprázdny reťazec nazývaný *axióm a* $P \subset V \times V^*$ je konečná množina pravidiel. Prevedenie $(a, \chi) \in P$ je možné zapísať ako *a*–> χ. Znak *a* a slovo χ sa nazýva *predchodca* a *následník*. Predpokladajme, že nejaké písmeno $a \in V$, kde najmenší jeden reťazec $\chi \in V^*$ je taký, že *a*–> χ. Ak žiaden produkt nie je explicitne špecifický pre daného predka $a \in V$, predpokladaný identický produkt *a –> a* patrí množine produktov *P*. OL – systém je *deterministický* (označovaný ako *DOL - systém*) len vtedy ak $a \in V$ kde je práve jeden $\chi \in V^*$ taký, že platí *a*–> χ.

Nech $\mu = a_1 \ldots a_m$ bude ľubovoľný reťazec nad *V*. Reťazec $\nu = \chi_1 \cdots \chi_m \in V^*$ je priamo odvodený (alebo generovaný pomocou) μ, značený jako $\mu \Rightarrow \nu$ len vtedy ak $a_i \rightarrow \chi_i$ pre všetky *i = 1*, ..., m. Reťazec ν je generovaný pomocou *G* s dĺžkou *n* ak existuje postupnosť reťazcov

$\mu_0, \mu_1, \ldots, \mu_n$ tak, že $\mu_0 = \omega, \mu_n = \nu$ a $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$.



Obrázok 4 - Rozvoj umelého vlákna (Anabaena catenula) použitím DOL – systémov

Nasledujúci príklad rozpisuje ďalšiu ilustráciu operácií, ktoré DOL systémy ponúka. Formalizmus sa používa na simulovanie rozvoja fragmentov v multibunkových vláknach takých, ktoré sa našli v modro-zelenej baktérii *Anabaena catenula* a rôznych riasach. Symbol *a* a *b* prezentuje cytologický stav buniek (ich veľkosť a pripravenosť k deleniu). Prípona *l* a *r* značí bunkovú polaritu, bude produkovaná špecifická pozícia s dcérskymi bunkami typu *a* a *b*. Rozvoj je popísaný nasledovným L – systémom.

$$\begin{aligned} &\omega : a_r \\ &p_1 : a_r \rightarrow a_l b_r \\ &p_2 : a_l \rightarrow a_l b_r \\ &p_3 : b_r \rightarrow a_r \\ &p_4 : b_l \rightarrow a_l \end{aligned} \qquad (2.1)$$

Začíname z jednoduchej bunky a_r (axióm), nasledujúca postupnosť reťazcov je generovaná takto:

$$a_r$$
$$a_l\, b_r$$
$$b_l\, a_r\, a_r$$
$$a_l\, a_l\, b_r\, a_l\, b_r$$
$$b_l\, a_r\, b_l\, a_r\, a_r\, b_l\, a_r\, a_r$$
$$\ldots \qquad (2.2)$$

Pod mikroskopom spozorujeme vlákno, ako postupnosť valcov s rôznou dĺžkou s *a* – typovými a potom b – typovými bunkami. Obrázok 4 - Rozvoj umelého vlákna (Anabaena catenula) použitím DOL – systémov.

2.3 Korytnačia interpretácia reťazcov

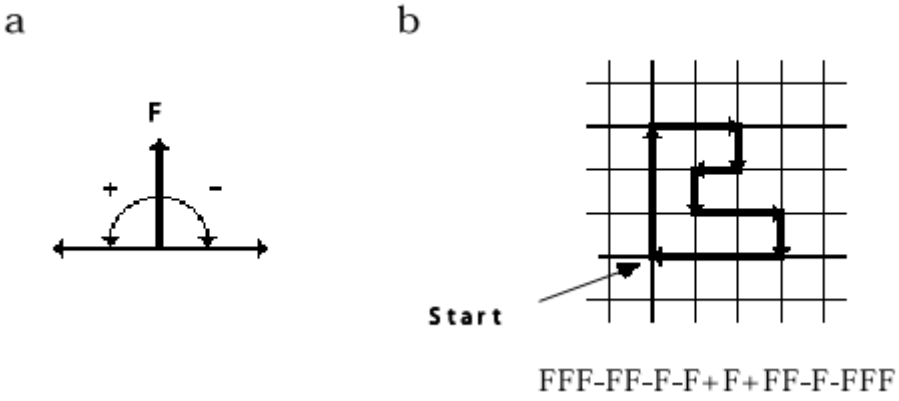
Najjednoduchším spôsobom ako znázorniť obrázky z *Anabaena catenulových* reťazcov je geometrickou interpretáciou. Písmenká z L – systémovej abecedy sú reprezentované graficky ako veľké alebo malé uhly s oblými rohmi. Generované štruktúry sú jedno-dimenzionálne reťazce uhlov, ktoré odrážajú postupnosť symbolov v príslušnom reťazci.

Aby sme mohli čo najlepšie modelovať rastliny je za potreby sofistikovaných grafických L – systémov. Prvým kto smeroval touto cestou je Frijters A Lindenmayer v roku 1974 a Hogeweg a Haper [4]. V obidvoch prípadoch boli použité predovšetkým L – systémy s deterministickou rozvetvenou topológiou modelov rastlín. Geometrické hľadiská ako dĺžka čiary alebo veľkosť uhla boli pridané až v po-spracovacej fáze. Práca Hogewega a Hespera bola následne rozšírená Smithom [15], ktorý demonštroval potenciál L – systémov pre použitie vytvárania realistického obrázku syntetík.

Szilard a Quinton [4]v roku 1979 navrhli odlišný prístup k L – systémom. Sústredili na predstavu, ktorá bola presne definovaná geometriou, ako napríklad kódovaný reťazec [4] a dokázali, že prekvapujúco jednoduchými DOL – systémami môžeme generovať fascinujúce, zložité krivky dnes známe ako *fraktály* [14]. Prusinkiewicz sa sústredil na interpretáciu základu pre LOGO – štýlovú korytnačku a prezentoval jej činnosť na príkladoch s fraktálmi a štruktúrami podobnými rastlinným modelom za použitia L – systémov.

Následne bude rozobratá základná myšlienka korytnačky. *Stav korytnačky* je definovaný ako trojica (x, y, α) , kde Karteziánske súradnice (*x, y*) určujú pozíciu korytnačky, a uhol α volaný

hlavička je prezentovaný ako smer, do ktorého sa korytnačka pozerá. *Veľkosť kroku d* a *uhol inkrementácie* δ,



Obrázok 5 - a) symboly, ktoré používa korytnačka *F, +, -. b) Prezentovanie reťazca. Uhol rovná 90stupňov. Na začitaku korytnačka smeruje hore.*

sú znázornené nasledujúcimi príkazmi:

F Pohyb dopredu o dĺžku *d*. Stav korytnačky sa mení na (x', y', α), kde $x' = x + d\cos\alpha$ a $y' = y + d\sin\alpha$. Nakreslenie čiary medzi bodmi (*x*, y) a (x', y').

F Dopredu o dĺžku *d* bez kreslenia čiary.

\+ Otočenie doľava o uhol δ. Ďalší stav korytnačky je $(x, y, \alpha + \delta)$. V smere ručičkových hodín.

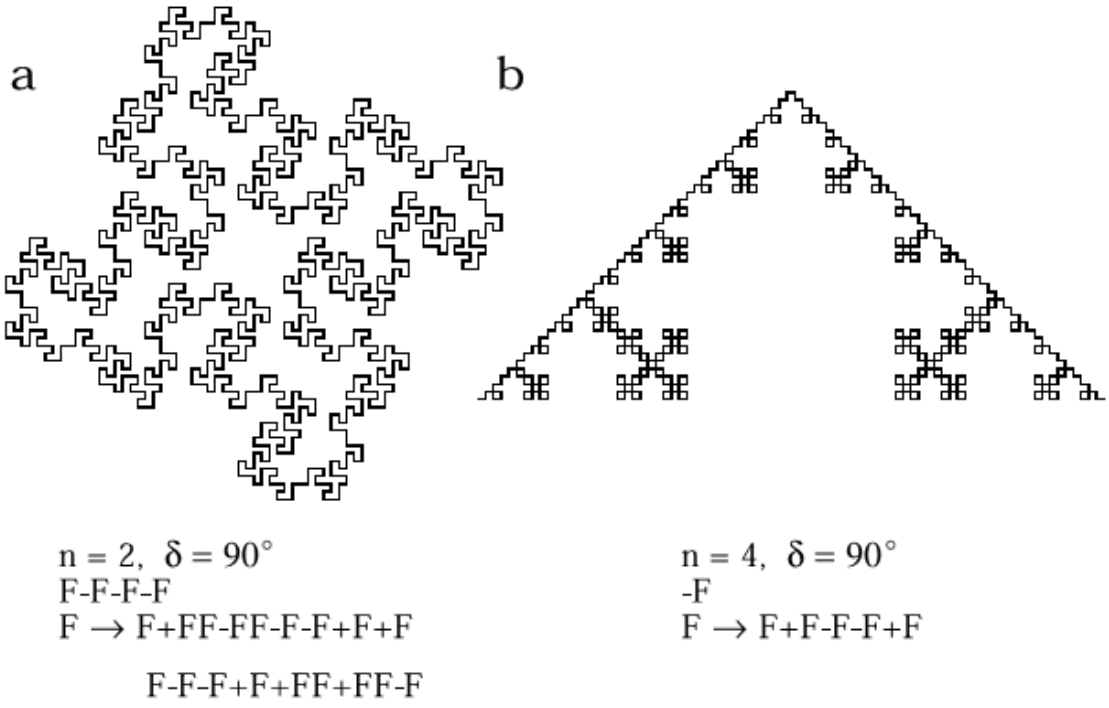
\- Otočenie doprava o uhol δ. Ďalší stav korytnačky je $(x, y, \alpha - \delta)$.

Dostaneme reťazec v, počiatočný stav korytnačky je (x_0,,y_o,α_0) s fixnými parametrami *d* a δ, vysvetlenie korytnačky z reťazca v je znázornené (pomocou množiny znakov) na obrázku. (Obrázok 5 - a) symboly, ktoré používa korytnačka *F, +, -. b) Prezentovanie reťazca. Uhol rovná 90stupňov. Na začitaku korytnačka smeruje hore.*. Táto metóda môže byť aplikovaná pomocou reťazcov, ktoré sú generované L – systémami. Napríklad v prílohe s obrázkami (Obrázok 21) sú ukázané štyri postupy pri tvorbe kvadratického Kochovho ostrova[4]. Obrázky sú generované L – systémami za pomoci týchto reťazcov:

$$\omega : F - F - F - F$$
$$p : F \rightarrow F - F + F + FF - F - F + F \qquad (2.3)$$

Obrázky znázorňujú odvodzovanie s dĺžkou od 0 do 3. Uhol δ rovná 90stupňov. Dĺžka kroku medzi jednotlivými obrázkami klesá štyrikrát.

Príklad nižšie prezrádza blízky vzťah medzi Kochovou konštrukciou a L –systémami. Inicializátor sa zhoduje s axiómom a generátor s nasledovníkom. Predok *F* znázorňuje jednoduchú hranu. L – systémy v tom prípade predstavujú kódovanie pre konštrukciu Kochoveho ostrova. Tu Obrázok 1sú ukázané príklady Kochovej krivky, ktoré môžeme generovať za použitia L – systémov. Menšia komplikácia môže nastať vtedy ak krivka nie je spojená. Druhú produkciu (za pomoci predchodcu *f*) je potrebné na udržanie vhodnej vzdialenosti medzi nimi. (Obrázok 6 - Príklad Kochovoej krivky generovanej pomocou L - systémov: a) Kvadratický Kohocov ostrov b) Kvadraticky modifikovaná snehová vločka) Ľahká modifikácia L – systémov ich robí veľmi vhodnými pre tvorbu nových Kochových kriviek. Obrázok 22 je nakreslených niekoľko modelov. Obrázok 23 znázorňuje postupné tvorenie Kochovej krivky.



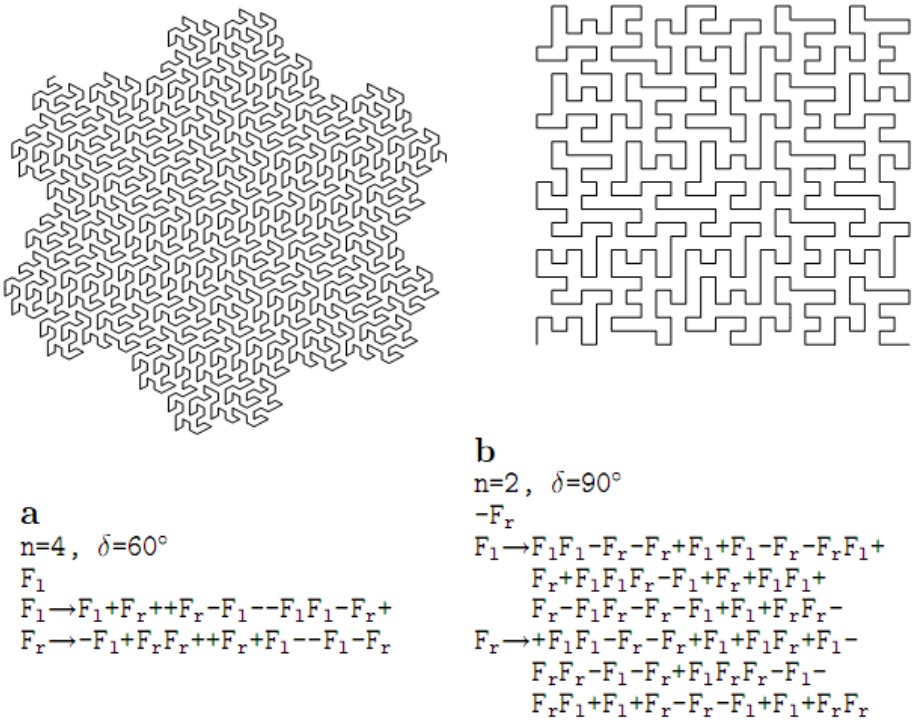
Obrázok 6 - Príklad Kochovoej krivky generovanej pomocou L - systémov: a) Kvadratický Kohocov ostrov b) Kvadraticky modifikovaná snehová vločka

2.4 Syntéza L – systémov

Náhodnou modifikáciou výsledkov z L – systémocv nám dávajú menšie pochopenie vo vzťahu medzi L – systémami a ich generovanými obrázkami. Avšak my si časť krát prajeme aby L – systémy, ktoré zachytávajú danú štruktúru alebo postupnosť štruktúr vyjadrovali proces rozvoja. V L – systémoch sa táto problematika nazýva *inferenčný problém* (inference problem). Preto sa zavádzajú nasledovné metódy, ktoré pomáhajú vytvoriť dané výsledky viac intuitívne a viac prirodzené. Sú to dva módy, ktoré sa spájajú s korytnačkou, volajú sa *prepisovanie hrán* a *prepisovanie uzlov.* V našom prípade pri prepisovaní hrán dochádza k substitúcii a hrana s polygonálnou stranou a pri prepisovaní uzlov dochádza k nahradeniu polygonálnym vrcholom.

2.4.1 Prepisovanie hrán

Prepisovanie hrán môžeme vidieť ako rozšírenú Kochovu konštrukciu. Vysvetlíme si to na príklade, ktorý je znázornený na Obrázok 7, kde sa nachádza Dračia krivka (dragon curve), ktorú sme vygenerovali za pomoci L – systémov. Obidva symboly F_l a F_r vytvárajú hranu, ktorá vzniká pomocou korytnačky a jej operácie *pohyb dopredu.* Substitúciou výsledkov F_l alebo F_r hranami pomocou párových čiar, ktoré sa otáčajú doľava alebo doprava. Veľa zaujímavých kriviek môže byť znázornených v prípade, že použije dva druhy hrán a to „vľavo“ a „vpravo“. Obrázok 7 demonštruje rozšírenia[4].

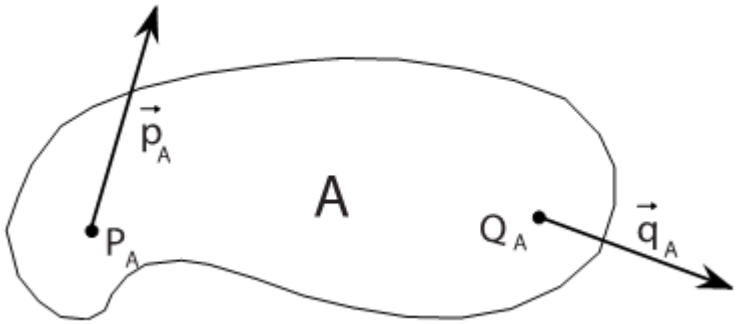


Obrázok 7 - ukážka FASS krivky pomocou generovania hrán a)hexagonálna Gosper krivka b) kvadratická Gosper krivka alebo E- krivka[3]

2.4.2 Prepisovanie uzlov

Hlavná myšlienka spočíva v substitúcií predošlých uzlov za nové polygóny. Aplikovanie je možné za pomoci korytnačky, pričom je rozšírená o symboly, ktoré reprezentujú ľubovoľný podobrazec ako je znázornené v prílohe (Obrázok 20) .Každý podobrazec *A* je z množiny podobrazca A ktoré sú prezentované:

- dvomi *dotykovými bodmi* ,ktoré sa označujú *vstupným bodom* P_a a *výstupným bodom* Q_a a
- dvomi *priamymi vektormi*, ,ktoré sa označujú *vstupným vektorom* $\vec{p}_A$ a *výstupným vektorom* $\vec{q}_A$.

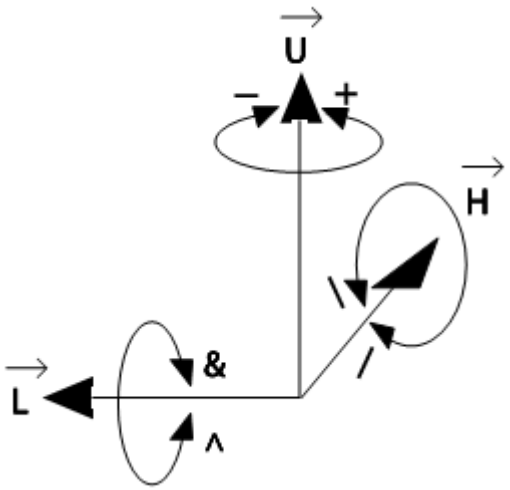


Obrázok 8 - opis podobrazca

Počas vysvetľovania označujeme reťazec v, symbol $A \in \mathrm{A}$. Na konci *A* je preložené a otočené podľa vstupného bodu P_A a smeru $\vec{p}_A$ s aktuálnou pozíciou a orientáciou. Natočené *A* je presunuté do výslednej pozície Q_A a *smeru* $\vec{q}_A$.

2.5 Modelovanie v trojrozmernom priestore

Korytnačia metóda môže byť rozšírená aby mohla pracovať v trojrozmernom priestore. Tento nápad uviedli Abelson a diSessa. Kľúč tejto myšlienky je v *orientácií* korytnačky v priestore za pomoci troch vektorov $\vec{H}, \vec{L}, \vec{U}$.



Obrázok 9 - Modelovanie v trojrozmernom priestore

Tieto vektory označujú kam sa korytnačka „pozerá“, smer „doľava“ a smer „dopredu“. Tento koncept predstavuje aktuálnu *orientáciu* korytnačky v priestore. Tieto jednotky majú jednotky dĺžky a sú závislými na sebe a sú odvodené od rovnice: $\vec{H} \times \vec{L} = \vec{U}$. Otáčanie korytnačky je vyjadrené pomocou tejto rovnice:

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

Kde **R** označuje maticu o veľkosti 3x3. Otáčanie pomocou uhlu α o vektory $\vec{H}, \vec{L}, \vec{U}$:

$$\mathbf{R}_U(\alpha) = \begin{bmatrix} \cos\alpha & \sin\beta & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

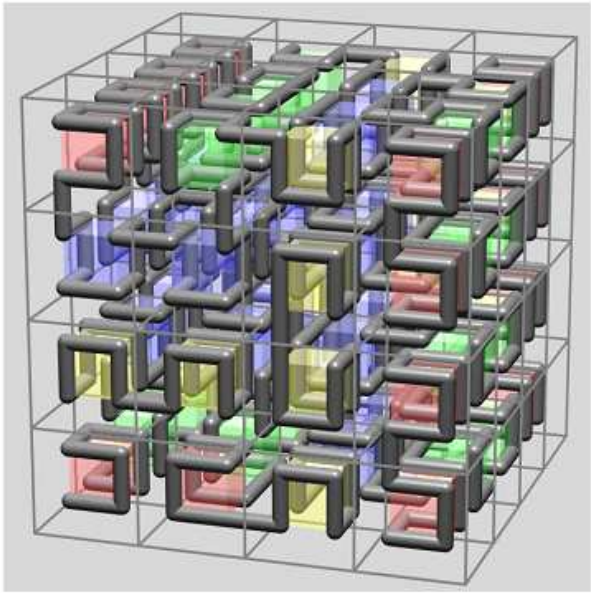
$$\mathbf{R}_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ -\sin\alpha & \cos\alpha & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$\mathbf{R}_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

(2.4)

Nasledujúce symboly ovládajú korytnačku v priestore (Obrázok 9)

+	Pre otočenie doľava o uhol δ s použitím rotačnej matice $\mathbf{R_U}(\delta)$
-	Pre otočenie doprava o uhol δ s použitím rotačnej matice $\mathbf{R_U}(-\delta)$
&	Klesnutie o uhol δ s použitím rotačnej matice $\mathbf{R_L}(\delta)$
^	Stúpanie o uhol δ s použitím rotačnej matice $\mathbf{R_L}(-\delta)$
\	Otočenie vľavo o uhol δ s použitím rotačnej matice $\mathbf{R_H}(\delta)$
/	Otočenie vpravo o uhol δ s použitím rotačnej matice $\mathbf{R_H}(-\delta)$
|	Otočenie dookola s použitím rotačnej matice $\mathbf{R_U}(180^O)$



n=2, δ=90°
A
A → B-F+CFC+F-D&F^D-F+&&CFC+F+B//
B → A&F^CFB^F^D^^-F-D^|F^B|FC^F^A//
C → |D^|F^B-F+C^F^A&&FA&F^C+F+B^F^D//
D → |CFB-F+B|FA&F^A&&FB-F+B|FC//

Obrázok 10 - Trojdimenzionálna rozšírená Hilberová krivka

Príklad použitia korytnačky v priestore za pomoci L – systému je znázornený v obrázku (Obrázok 10.) Kde L – systém bol vytvorený na základe výmeny uzlov a s použitím *macrocubec* namiesto *macrotiles.*

3 Syntéza 3D scény

3.1 Generovanie mesta a ulíc

Cieľom bude vygenerovať subsystém ciest, povrchu a domov tak, aby vytvárali mestské prostredie. Úsilím je vytvoriť daný systém tak, aby výsledný produkt pôsobil čo najreálnejším dojmom.

Základným kameňom pri návrhu tohto systému bola L gramatika. V podstate by sa mohol rozdeliť celý systém na tri hlavné časti. Prvou časťou je generovanie reťazca, analýza vygenerovaného reťazca a následné vykresľovanie objektov.

3.2 Generovanie reťazca

Prvým článkom generovania je tvorenie reťazcov, v ktorých sú použité DOL- systémy a prepisovacie pravidlá. Tento reťazec je tvorený pomocou *inicializátora*, ktorý rozširuje *generátor*.

Úlohou reťazca je vytvoriť základnú štruktúru, ktorá bude hlavným podkladom pre tvorbu mestskej infraštruktúry. Spočiatku sa jednalo hlavne o generovanie infraštruktúry ciest, ale následne nato boli vhodným základom aj pre umiestňovanie domov.

Systém vytvárania jednotlivých symbolov reťazca je podobný korytnačej interpretácii. Jedná sa o modifikovanú verziu, pričom symboly, ktoré sa nachádzajú v reťazci majú takúto významovú hodnotu:

A	kreslenie cesty určenou dĺžkou
Y	kreslenie cesty s domom s určenou dĺžkou
+	zmena uhlu o určený uhol doľava
-	zmena uhlu o určený uhol doprava

Pomocou týchto základných operácií môžeme vygenerovať infraštruktúru mesta. Činnosťou tohto procesu je zväčšovať generovaný reťazec pomocou definovaných pravidiel. Tu sa využíva prepisovanie reťazcov. Pravidlá môžeme vytvoriť podľa toho ako chceme aby náš výsledný produkt vyzeral. Inými slovami pravidlá A a Y prestavujú nastaviteľné generátory.

Ďalším dôležitým faktorom pri stavbe reťazca je *úroveň generácie,* tá udáva do akej „hĺbky“, alebo rozvetvenia má byť dané generovanie uplatnené. Uvedieme si to na jednoduchom príklade kedy:

Generácia = 1

Inicializátor ω: *A*

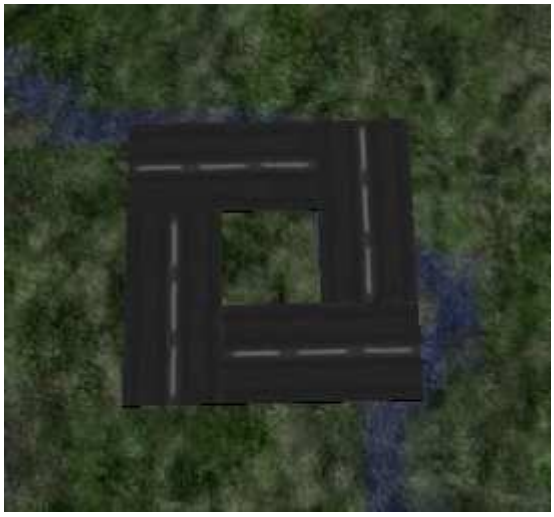
A: *A+A+A+A*

Y: pre jednoduchosť Y nebude do generovania zasahovať

Výsledkom budereťazec

A+A+A+A

Obrázok 11je prezentuje grafickú prezentáciu reťazca.



Obrázok 11- výsledný reťazec

3.2.1 Analýza vygenerovaného reťazca

Ďalším aplikačným článkom pri tvorbe krajinky je *analýza vygenerovaného reťazca* ktorý prijíma na vstupe reťazec, ktorý postupne prechádza symbol po symbole a vypočítava začiatočné a koncové body pre výsledný kreslený objekt. A tiež natavuje uhol otočenia daného objektu, v tejto časti sme schopný ovplyvňovať dĺžku jednotlivých objektov, ich jednotlivé odsadenie od seba, a jednu z najdôležitejších úloh, ktorou je určiť či sa jednotlivé objekty navzájom nepretínajú, alebo neprekresľujú cez seba.

3.3 Java Monkey Engine

Java Moneky Engine (ďalej len jME) je rozhranie, ktoré je použité pre vykresľovanie objektov, kvôli tomu aby výsledný obraz bol v priestorovom zobrazení. jME bolo navrhnuté ako rýchlo realistický realtimový grafický engine. S výhodami, ktorými disponuje grafický hardware počítača a programovacím jazykom *Java* je zrejmá potreba knižníc javy . jME preferuje vysokým výkonom pre vykresľovanie scény. Je kompatibilná s *OpenGl,* ktoré obsahujú mnohé moderné grafické karty a uľahčujú tak prácu návrhárom hier. jME bola navrhnutá s jednoduchosťou používania mysle pri súčasnom zachovaní výkonu moderných grafickým programov. Modulárna konštrukcia zaisťuje, že ako sa výkon hardwaru grafiky zlepšuje tak sa OpenGL snaží držať tempo a vyhovieť mu, tým pádom jME bude schopné rýchlo využívať najnovšie funkcie.

V popredí jME je graf scény. Scéna je dátová štruktúra, ktorá udržiava dáta zo sveta. Herné dáta sú udržiavané v stromovej štruktúre, listové uzly reprezentujú základné herné prvky. Tieto základné prvky sú zvyčajne poskytované scéne, alebo sú prehrávané cez zvukové karty. Organizácie scény je veľmi dôležité a zvyčajne závisí na konkrétnej aplikácií. Avšak typická scéna zastupuje veľké množstvo dát, ktoré boli rozdelené do menších ľahko udržiavaných kusov. Spravidla platí, že tieto kúsky sú zoskupené do určitých vzťahov. Pokiaľ nie sú potrebné tieto zoskupenia na aktuálnej scéne, je možné ich odobrať. Tým, že ich odoberme, nie je potrebné ich spracovať, a CPU s GPU ušetrí čas, ktorý by strávil zaoberaním sa týmito údajmi, čo vedie k tomu, že rýchlosť hry sa zlepšuje.

Kým scéna je jadrom pre grafické prvky jME, používanie nástrojov poskytuje prostriedky pre rýchle vytvorenie grafického prostredia. Pre vytvorenie okna, vstupného systému, kamerového systému a herného systému použijeme zavolanie nie viac ako jednej, maximálne dvoch metód. To umožňuje programátorovi zastaviť plytvanie času pri riešení systému a ponúka viac času v tvorení konkrétnej aplikácií.

Displej a render systému poskytujú abstrakciu pre komunikáciu s grafickou kartou. Hlavné herné slučky sú poskytované s cieľom umožniť jednoduché rozšírenie zo slučkových tried a jednoduchý vývod aplikácií.

Skutočné vykresľovanie scény grafu je oddelené od užívateľa. Čo prináša výhodu v možnosti použiť swapovanie a preto je možné zobrazovať scénu grafu na iných systémoch bez porušenia jediného riadku kódu aplikácie. Napríklad ak spustíte pomocou *LWJGL* od Puppy Games, tak môžete ľahko prejsť na *JOGL* od Sun, pretože rozdiel medzi nimi je minimálny, nemusíte prerábať celý projekt. [6]

jME prináša výhody, ktoré pomáhajú vytvoriť aplikáciu čo najviac použiteľnú. Ponúka triedy, ktoré majú významné postavenie pre tvorbu grafických aplikácií. My však využívame triedu *BaseGame*.

Je to trieda, ktorá pomáha vytvoriť prostredie do ktorého bude náš model mesta vkladaný. Ponúka aj možnosť pohybu v danom teréne, teda ide o akúsi virtuálnu prechádzku prvej osoby.

3.4 Vykresľovanie ciest

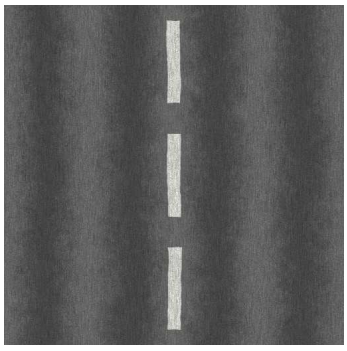
Cesty sú kreslené na základe informácií o začiatočných a koncových bodoch, ktoré zistíme z analýzy. Cesta je prezentovaná pomocou jME objektu *box*[6]. Box (debna)je generovaný na základe týchto kritérií:

- Minimálny bod
- maximálny bod.

Všetkých osem bodov z ktorých sa box skladá je tvorených za pomoci týchto dvoch bodov. Principiálne to funguje tak, že dostaneme bod označovaný *Min_P*, ktorému sú priradené začiatočné súradnice, tie získame z *analýzy reťazca.* A druhý bod označovaný *Max_P*, ktorý takisto vygeneruje *analýza reťazca.* Ukážeme si na príklade, kde sú zadané body vložené do trojrozmerného súradnicového priestoru.

```
Min_P 0, 1, 0
Max_P 10,1,10
```

Týmto spôsobom vytvoríme objekt *box* s veľkosťou 10. Lenže cieľom je vytvoriť objekt, ktorý má stelesňovať cestu a to dosiahneme tým, že danému objektu priradíme *textúru*[7], ktorá vytvorí objekt podobajúci sa reálnej ceste. Textúru, ktorú sme použili je na obrázku.(Obrázok 12).



Obrázok 12 – Textúra cesty

A celkový výsledok (Obrázok 13).

Obrázok 13 - Výsledná cesta

Pri tejto aplikácií nastal problém s vykresľovaním *Box*, pretože *analýza reťazca* pracuje tak, že vždy nanovo prepočítava body a keď sa pokúšame nakresliť objekt s tým, že *Min_P* má väčšie hodnoty ako *Max_P*, teda v tom prípade kreslíme objekt opačným smerom a to zhora dole, nastával problém v tom, že daný objekt nevedel jME správne zobraziť. Preto musíme v analýze prepočítať tieto hodnoty. To sme robili tak, že sme zistili hodnoty bodov a tie sme porovnávali.
Príklad:

Min_P = 10, 1, 10

Max_P = 0, 0, 0

Pre nás ma podstatnú úlohu súradnica *x* a *z*, tie určujú smer kreslenia v jednej rovine.

1. Ak $Min\_P_A > Max\_P_A$ tak potom vieme povedať, že objekt je nutné otočiť tzn. vymeniť súradnice. A cesta je kreslená normálnym spôsobom len s tým, že spomínané body sú vymenené
2. Ak $Min\_P_Z > Max\_P_Z$ tento krok je obdobou predošlého kroku, len sa jedná o *z* súradnicu

Pri generovaní „dvoch ciest“ sa vytvárajú dva boxy. Uvedieme názorný príklad:

Reťazec ω: AA

Čo znamená, že chceme zakresliť dve cesty za sebou. Prebehne to nasledovným spôsobom:
Prebehne *analýza reťazca*, vykonajú sa dva behy. V prvom sa vygenerujú súradnice *Min_P* a *Max_P* ktoré vytvoria prvý box, a v druhom chode sa dopočítajú súradnice pre druhý objekt, čiže dochádza k spájaniu cesty čo vytvára dlhšiu cestu.

V *analýze reťazca* je riešená aj rotácia týchto objektov, ktorá nastáva pri načítaní symbolu „+“. Vtedy sa počítajú hraničné body tak, že sa zmenia jednotlivé veľkosti súradníc tak, že výsledný obdĺžnik je kreslený vodorovne, vtedy sa k nemu pripája textúra, ktorá je v súhlasnom smere.

3.5 Vykresľovanie domov

Vykresľovanie domov je možné na základe informácie, ktorá hovorí o tom kde sa má daný objekt nachádzať.

V tom to prípade dochádza k vykresľovaniu obdobným spôsobom ako bolo pri kresbe ciest s tým rozdielom, že jednotlivé body, ktoré značia to kde sa majú nachádzať budovy sú postavené na základe polohy cesty. Čiže sa nejdená len o vykreslenie budovy ale je k nemu prikreslená aj cesta aby bol výsledok realistický.

Ako budovy je možné použiť ľubovoľné grafické formáty, ktoré obsahujú modely domov. V tomto prípade používame *obj formát*[8]. V podstate importujeme tento model za pomoci funkcií jME a ukladáme ho do vygenerovanej pozície.
Nasledujúci príklad je potrebné uviesť,aby bolo jasné, že to kedy bude generovaný objekt je spôsobené pravidlom *Y*.

Reťazec $\omega : Y$

Teraz sa vygeneruje cesta s vypočítaným bodom tak aby bol objekt budovy umiestnený vedľa nej.(Obrázok 14).

Obrázok 14 - Vkladanie budovy

3.6 Vykresľovanie terénu a pozadia

V tomto kroku sa vytvára základný povrch na ktorý sú potom umiestňované objekty. Na obrázku 16 je ilustračný obrázok ukazujúci terén.

Terén je generovaný na základe výškovej mapy [5]. Vhodnou voľbou výškovej mapy a jej editovaním vznikne povrch, ktorý je použiteľný pre náš prípad. Implementujeme ju za pomoci jME s triedou BaseGame. Zhotovíme takú mapu, aby poskytovala čo najrovnejší povrch. Obrázok 15 ukazuje použitú textúru.

Obrázok 15 - Výšková mapa

Mapa sa prevedie na výsledný terén (Obrázok 16)

Obrázok 16 - Terén

3.7 Generovanie stromov

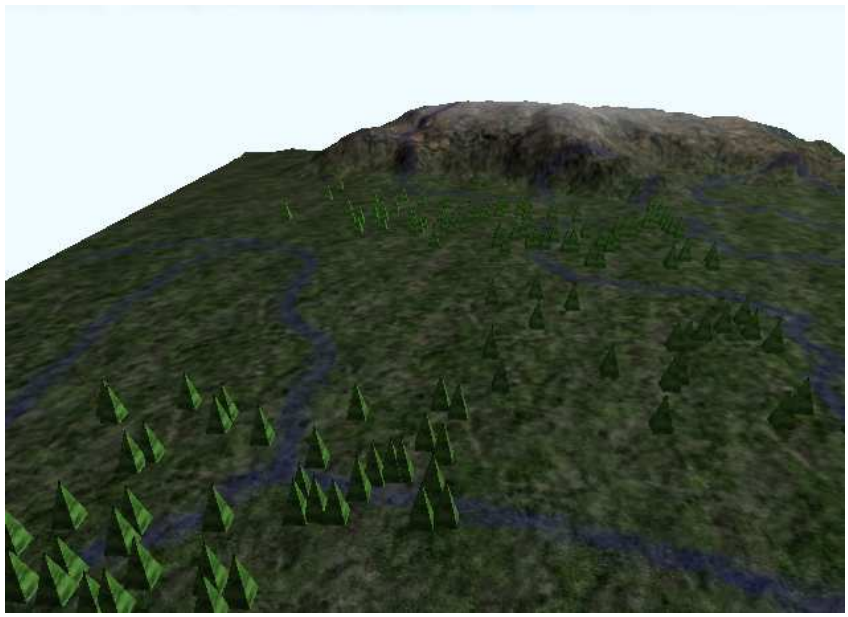
Na generovanie stromov použijeme jME a vytvorenú metódu náhodného generovania lesa stromov.

Na stelesnenie stromu sme použili objekt *pyramid*[9] na ktorú pripevníme textúru stromu, výsledkom je objekt podobajúci si na strom.

Metodika vytvárania lesov by sa dala opísať nasledovným postupom:

1. Vytvoríme strom, ktorý pomocou generátora náhodných čísel umiestnim., V generátore je nastavený malý rozptyl (tj. je nastavený malý interval čísel), preto aby vznikali zhluky stromov, ktoré vytvárajú týmto spôsobom les.
2. Krok 1 zopakujeme niekoľkokrát, tak aby bol počet stromov v takom množstve aby vytvárali ilúziu lesu.
3. Tento zhluk stromov generujeme niekoľk krát, vznikajú skupinky stromov, ktoré umiestňujeme na scénu za pomoci generátora náhodných čísel, ktorý má však väčší rozptyl (je nastavený veľký interval čísel aby stromy mohli byť uložené kdekoľvek na scéne).
4. Krok 3 zopakujeme niekoľkokrát tj. lesov bude viacej.

Jednotlivé zhluky stromov, teda lesy ukladáme do triedy *spatial*[10] v ktorej sa môžu nachádzať takéto typy objektov.



Obrázok 17 – Lesy

3.8 Prekrývanie objektov

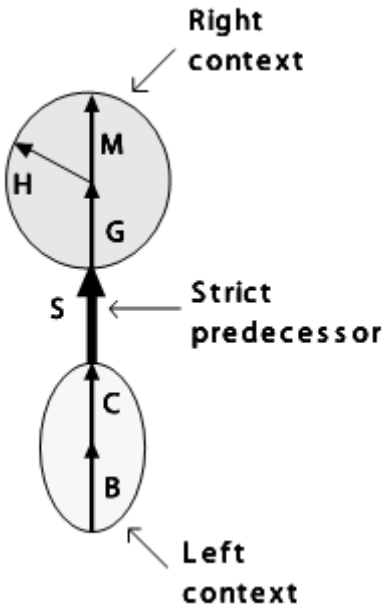
Ďalšou problematikou bolo vytvoriť objekty tak ,aby sa navzájom neprekrývali, a teda dávali zmysluplný dizajn. Tu uvažujeme medzi dvomi rozdielnymi metódami:

- Pomocou kontextovej podmienky v L - systémoch
- Pomocou java Monkey Enginu

3.8.1 Kontextové L – systémy

Produkty v DOL – systémoch sú bezkontextové, tj. aplikované bez závislosti na kontexte v ktorom sa nachádza predchodca. Avšak, je možné aby bol produkt aplikovaný na kontexte predchodcu. Tento efekt je užitočný pri [3] iteráciách v problematike prekrývania objektov. Najvhodnejšie bude, keď si to vysvetlíme na vykresľovaní cesty tak aby sa navzájom neprekrýkali. *2L – systémy* používajú nasledovnú formu: $a_l < a > a_r \rightarrow \chi$ kde *a* (volaný priamy predchodca) môže byť produkovaný reťazcom χ za predpokladu len vtedy ak *a* za symbolom a_l nasleduje a_r. Symboly a_l a a_r je ľavá forma ľavej strany a pravá strana je kontextom z *a.* 1L – systémy majú len jednostranný kontext.

Využitie si popíšeme na príklade:



Obrázok 18 - a) Predchodca z kontextového stromu

Vstupný reťazec ω: *baaaaaaaa*

p_1: *b <a –> b*

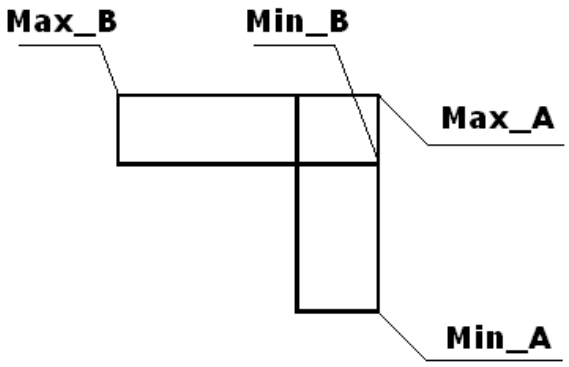
p_2: *b –> a*

Niekoľko prvých vygenerovaných reťazcov:

baaaaaaaa
abaaaaaaa
aabaaaaaa
aaabaaaaa
aaaabaaaa
...

Písmeno *b* sa v reťazci pohybuje z ľavej strany doprava. [3]

Bolo to jedno z uvažovaných možných riešení, ktorýho výsledkom malo byť odstránenie vlastnosti prekrývania objektov. Spočíva to v podmienke kedy by sa dalo určiť kedy dochádza k prekrývaniu objektov. To by sa dalo využiť pri vstupnom reťazci ω: *A+A*, vtedy by vznikalo zalomenie cesty. Najprv by bola nakreslená prvá cesta, tá by mala svoje počiatočné a koncové body, môžeme ich označiť *Min_A* a *Max_A*, v *analýze reťazca* by sa pri načítaniu symbolu „+“ zmenil uhol vykreslenia a dochádza ku kresleniu druhej cesty, tá má body *Min_B* a *Max_B*. Teraz vzniká moment kedy nastáva problém (Obrázok 19). Úlohou je zamedziť prekrývaniu ciest.



Obrázok 19 - Prekrývanie cist

Bolo by to možné za pomoci týchto pravidiel:

Vstupný reťazec ω: *A+A*
p_1: *A <+ –> C*

Kedy *C* nahradí dve cesty jednou cestou, ktorá by prezentovala roh (zákrutu).
Táto problematika však bola vyriešená v *analýze reťazca*, iným spôsobom. Využilo sa to, že objekt reprezentujúci cestu teda *box* vieme modifikovať a dokážeme zmeniť jeho tvar a veľkosť.

Takže najprv vykreslíme prvú cestu, na základe porovnávania *Min* a *Max* bodov dokážeme určiť akým smerom bude predmet vykresľovaný a pri kreslení druhej cesty posunieme súradnice začiatočného bodu tak, aby sa nachádzal mimo oblasti prvej cesty.

3.8.2 Java Monkey Engine

Tento spôsob je použitý pri prekresľovaní objektov, teda v situácii ,kedy sa jeden objekt, napríklad dom s druhým domom vykreslia cez seba. To je samozrejmé, že toto je nežiaduca situácia, ktorú treba eliminovať. Tento problém je dôsledkom, ktorý vzniká pri generovaní reťazca. Reťazec ktorý vzniká je tvorený generáciami, a teda nie je jasné, kde sa budú dané objekty vykresľovať a či sa nepretínajú.

Takže problém bol riešený za použitia funkcií, ktoré nám poskytuje jME rozhranie. Vtedy jednotlivým objektom pridelíme ich *boundny* (hranice objektu) [11], ktoré „zabalia“ celý objekt a vytvoria okolo neho hranice, ktoré môžeme detegovať a dokážeme zistiť či sa dané objekty navzájom pretínajú a na základe toho rozhodnúť, ktoré objekty budú vykreslené.

3.8.3 Grafické rozhranie

Pre zadávanie vstupného reťazca použijeme navrhnuté gui, ktoré je vytvorené v swing[12] je to Java nástroj ktorý vyvinul Sun Microsystem.

V tomto gui môžeme zadať vstupný reťazec, pravidlá, úroveň generácie a následne vykresliť krajinku. Ovládanie je intuitívne, dôležité je však zadať vhodný vstupný reťazec.

3.8.4 „Chôdza“ po krajinke

V tomto prípade sa použijeme jME rozhranie s triedou *BaseGame*[12]. Obraz je z pohľadu prvej osoby.

Tabulka 1 - ovládanie

Tlačidlo	W	S	A	D	Q	Z	Hore	Dole	Vľavo
Akcia	Pohyb dopredu	Pohyb dozadu	Pohyb doprava	Pohyb doľava	Zdvih	Klesnutie	Pohľad hore	Pohľad dole	Otočenie vľavo

Vpravo	T	L	C	B	F1	N
Otočenie vpravo	Wireframe mód	Svetlo	Výpis pozície kamery	Zap/Vyp zobrazovania hraníc	Fotka	Normály

4 Záver

Táto práca mi poskytla možnosť pracovať s novými nástrojmi, ktoré som využil v riešení môjho zdania. Z použitých nástrojov sú to: implementačný jazyk Java, rozhranie jME, Swingovský nástroj pretvorbu grafického rozhrania a Lindmayerové systémy. Štúdium týchto nástrojov mi rozšírilo obzor v rovine teoretickej (ako sú práca s reťazcami, symbolmi, konečnými automatmi a abecedami) aj praktickej. Predtým som nemal skúsenosti s aplikáciami zameranými na grafické zobrazovanie, preto táto práca mala veľký význam v tejto tematike.

Moja práca môže slúžiť ako podklad pre ďalšie využitie L-systémov v grafike.

Ďalší vývoj

Ďalším vývojom projektu by mohlo byť zavedenie nových objektov(ako napríklad budovy, križovatky, lampy, mosty), ktoré by dodali mestu rozmanitosť. Tiež si myslím, že by bolo možné použiť *rozptyľovaciu funkciu*, ktorá by umiestňovala väčšie domy do stredu mesta, vytváral by sa tak *metropolitný model mesta.* Ďalším vhodným doplnkom by bolo vytvoriť funkciu, ktorá by dokázala vyexportovať vygenerovaný terén do známych grafických formátov, ktoré by bolo možné spracovať v grafických aplikáciách (akým je napr. Blender). Zaujímavým vylepšením by bolo vytvárať aj budovy pomocou L – systémov, zabránilo by sa tak problému s načítavaním objektov a hlavným problémom, ktorý spočíva v ťažkosti nachádzanie objektov vhodných pre importovanie takých, ktoré sú voľné šíriteľné.

Určite vhodným doplnkom by bolo zavedenie istej miery náhodnosti do systému, vykresľované mesto by vyzeralo tak po každom vykreslení trocha inak ale jadro by sa zachovalo.

Ak by sme chceli urýchliť proces výpočtu a vykresľovania scény, vhodným krokom by bolo použiť iný implementačný jazyk.

Túto prácu som riešil dekompozičnou metódou, snažil som sa projekt rozdeliť na menšie celky a tie potom konštruovať, vznikali tak malé programy, ktoré vykonávali jednoduché činnosti.

Podľa výsledkov, ktoré som dosiahol som zistil, že metóda, ktorá bola použitá pri tvorbe krajinky je vhodnejšia na generáciu rastlín, alebo pre prezentovanie vlastností Lindenmayerových systémov. Pre tvorbu krajinky je vhodnejšie použiť L- systémy spojené s vyhľadávaním najbližšej cesty[14]. Nevýhody L- systémov spočívajú v ich implementácií, pretože je zložitá. Výhodou je to, že dokážu výsledný produkt vytvoriť rýchlejšie[15].

Ďalšou dôležitou vecou čo si je treba uvedomiť je, že L – systémy sa vytvárajú generovaním a postupným nahradzovaním symbolov v reťazci, čím vyššia je úroveň generácie a väčší počet pravidiel tak tým dlhšie trvá toto nahradzovanie, čo je to náročná aplikácia pre CPU. V mojej práci je potrebné vždy zvoliť správne pravidlá a vstupný reťazec, aby vykresľovaná krajinka bola čo najreálnejšia.

Literatúra

[1] R. Smith. Plants, *fractals, and formal languages*. Výťah z SIGGRAPH ’84 Minneapolis, Minnesota, July 22-27, 1984) in Computer Graphics, 18, 3 (July 1984), strany 1–10, ACM SIGGRAPH, New York, 1984

[2] *Pacsal Muellers Wiki*, Pascal Mueller. 2009. Dostupné na URL: <http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperCities>

[3] Przemyslaw Prusinkiwicz and arktid Lindenmayer,*The Algorichmic Beauty of Plants*, New York, 1990 [online] Dostupné na URL: <http://algorithmicbotany.org/papers/>

[4] D. Frijters and A. Lindenmayer. *A model for the growth and flowering of Aster novae-angliae on the basis of table (1,0)L-systems*. In G. Rozenberg and A. Salomaa, editors, L Systems,Lecture Notes in Computer Science 15, Berlin, 1974.

[5] *POV- Ray*: Documentation. [online] Dostupné na URL: <http://www.povray.org/documentation/view/3.6.1/279/>

[6] *jME Wiki* : Box. 2007. [online] Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=box>

[7] http://www.jmonkeyengine.com/wiki/doku.php?id=texturerenderer

[8] *OBJ Files – A 3D Object Format*. 2008. [online]. Dostupné na URL: <http://people.sc.fsu.edu/~burkardt/data/obj/obj.html>

[9] jME Wiki : *pyramid*. 2007. [online] Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=pyramid>

[10] jME Wiki : *spatial*. 2007. [online] Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=spatial>

[11] jME Wiki : *bouding* volume. 2007. [online] Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=boundingvolume>

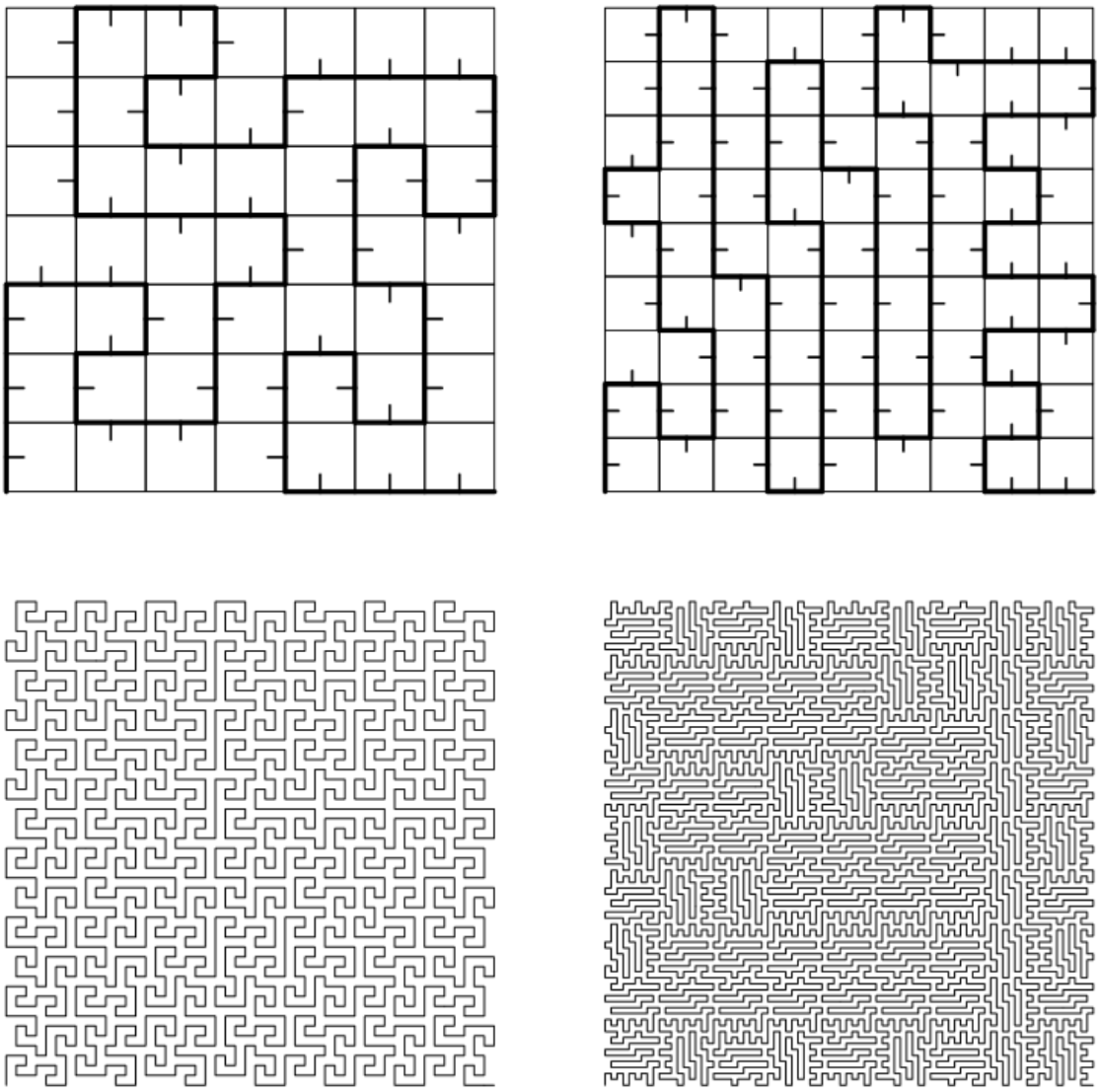
[12] jME Wiki : *pyramid*. 2007. [online] Dostupné na URL: <http://www.jmonkeyengine.com/wiki/doku.php?id=simplegame>

[13] *The Java Tutorials* 2008. [online]. Dostupné na URL: <http://java.sun.com/docs/books/tutorial/uiswing/>

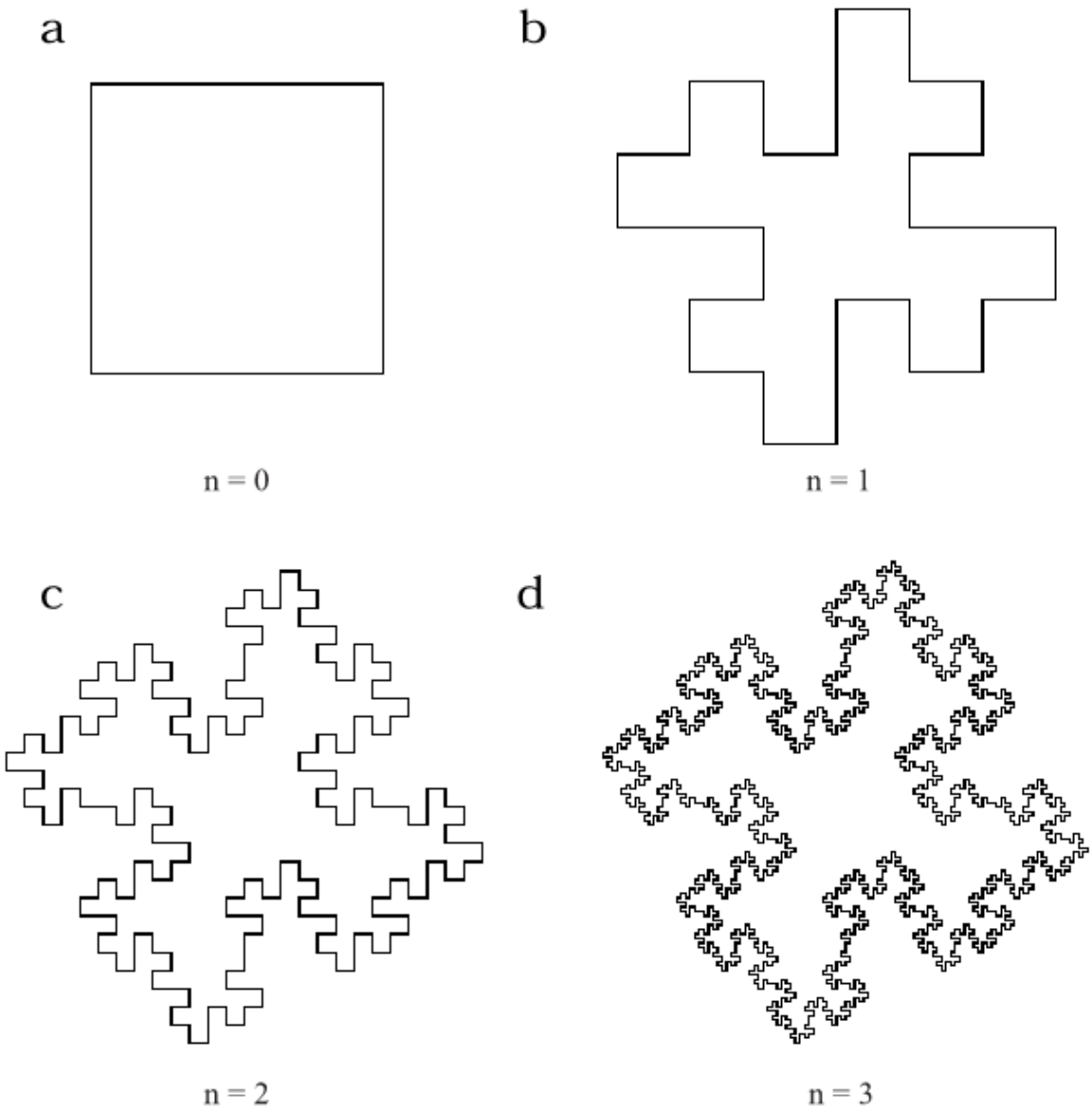
[14] M. F. *Barnsley. Fractals everywhere. Academic Press, San Diego,* 1988.

Zoznam príloh

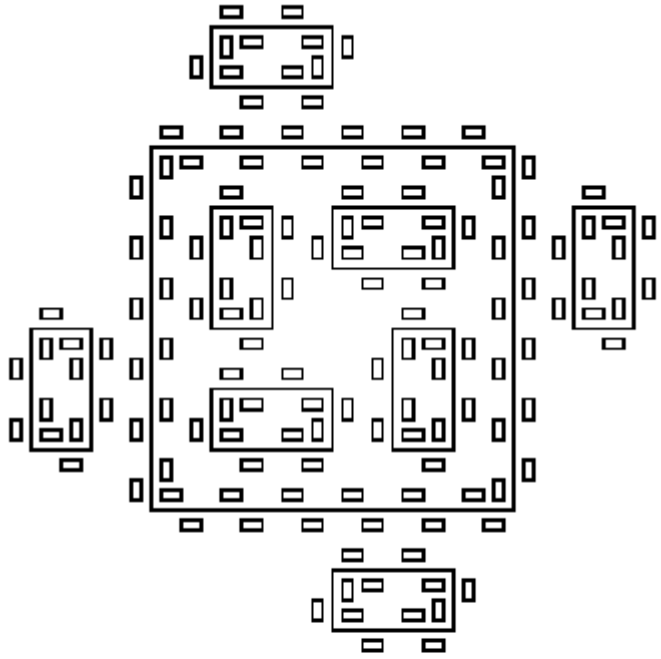
Príloha obrázkov



Obrázok 20 - Príkad FASS kriviek je generovaný štvorcou mrežou použitím prepisovania hran a) SqauRecurve (size 7x7) b) E-tour (size 9x9)



Obrázok 21 - Generovanie kvadratického Kochovho ostrova



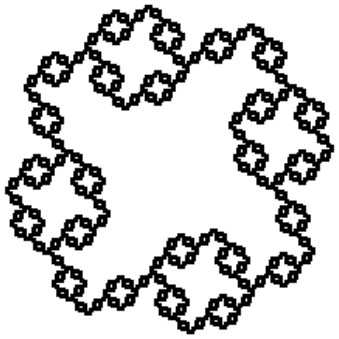
$n = 2, \ \delta = 90^\circ$

F+F+F+F

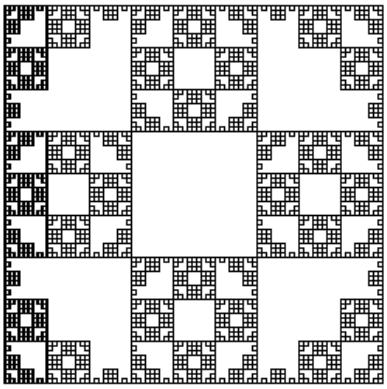
F → F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF

f → ffffff

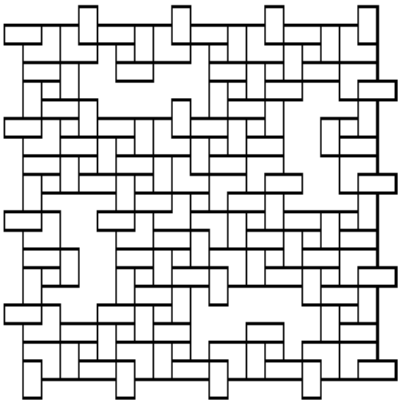
Obrázok 22 - Kombinácia ostrovu a jazera



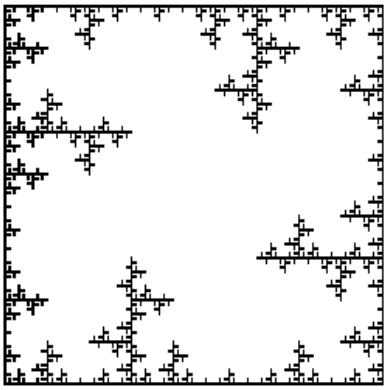
a $n = 4,\ \delta = 90°$
F-F-F-F
$F \rightarrow FF\text{-}F\text{-}F\text{-}F\text{-}F\text{-}F+F$



b $n = 4,\ \delta = 90°$
F-F-F-F
$F \rightarrow FF\text{-}F\text{-}F\text{-}F\text{-}FF$

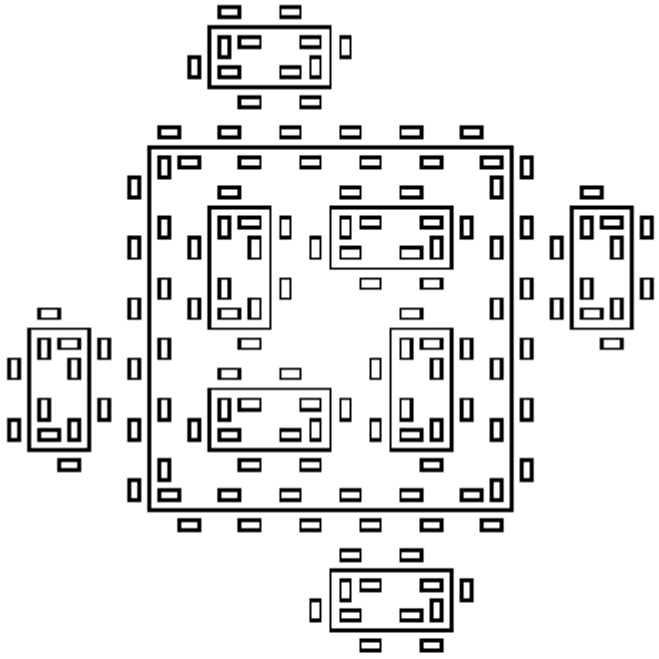


c $n = 3,\ \delta = 90°$
F-F-F-F
$F \rightarrow FF\text{-}F+F\text{-}F\text{-}FF$



d $n = 4,\ \delta = 90°$
F-F-F-F
$F \rightarrow FF\text{-}F\text{--}F\text{-}F$

Obrázok 23 - Postupné tvorenie Kochovej krviky



$n = 2,\ \delta = 90^\circ$

F+F+F+F

$F \rightarrow$ F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF

$f \rightarrow$ ffffff

Obrázok 24 - Kombinácia ostrovu a jazera

Príloha príklad použitia

Tu sa bude nachádzať príklad použitia vytvoreného projektu s jeho grafickým rozhraním

Grafické rozhranie

V gui môžeme zadať vstupný reťazec z ktorý slúži ako hlavný kameň vykresľovania, pravidlá A a Y, pravidlo A určuje ako bude generovaná cesta, pravidlo Y určuje ako budú generované budovy. Ďalším parametrom je hĺbka generácia, ktorá udáva do akej úrovne má byť reťazec generovaný.

Obrázek 25 – Gui

Scéna

Obrázek 26 – scéna

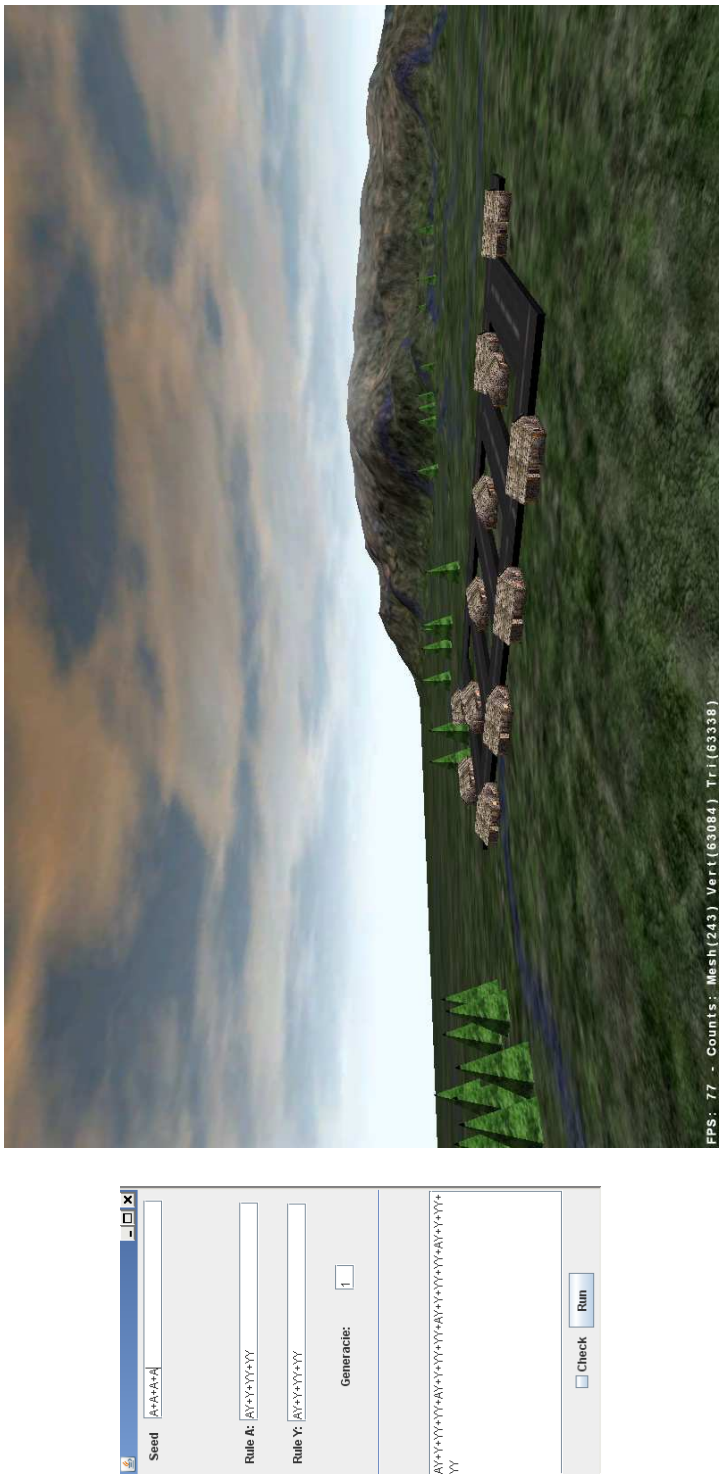
Príklady

Ako vstupný reťazec je použitý: A-Y–Y-Y

Pravidlo A: AY+Y+A+AY

Počet generácií: 2

Po stlačení tlačidla „Run“ nastane vykresľovanie scény v novom okne.



Obrázek 27 - príklad použitia