L-systemy

dokumentacja do projektu z przedmiotu „Obliczenia Symboliczne II”

Marcin Rociek
Informatyka, III rok

14 stycznia 2001

Spis treści

1 L-systemy

L-systemy (systemy Lindenmayera) zostały opracowane w 1968 roku przez biologa Aristida Lindenmayera. Umożliwiają modelowanie biologicznego wzrostu za pomocą przepisywania ciągów znaków. Przepisywanie jest techniką polegającą na sukcesywnym zamienianiu części prostego początkowego ciągu znaków (aksjomatu) zgodnie z ustalonym zbiorem reguł przepisywania (produkcji).

Późniejsze prace nad L-systemami umożliwiły wykorzystanie ich m.in. do modelowania bardziej złożonych organizmów i tworzenia realistycznej grafiki komputerowej przedstawiającej rośliny, krzewy, drzewa itp.

1.1 Rodzaje L-systemów

1.1.1 L-systemy deterministyczne, bezkontekstowe

L-Systemy deterministyczne i bezkontekstowe (*D0L-systems*) są najprostszą klasą L-systemów. Składają się one z ustalonego słowa początkowego (aksjomatu) oraz zbioru reguł rozrostu (produkcji). Produkcje są postaci $a \to \chi$, co oznacza że litera a ma być zamieniona na pewne słowo χ. Jeśli jakiś znak nie ma przyporządkowanej produkcji, stosowana jest produkcja domyślna, tj. $a \to a$.

Przykładowo weźmy pod uwagę ciągi znaków składających się tylko i wyłącznie z liter a i b, gdzie każda z tych liter może w danym ciągu występować dowolną ilość razy. Z każdą literą skojarzona jest odpowiednia reguła rozrostu (produkcja). Reguła $a \to ab$ mówi, że litera a ma być zastąpiona ciągiem ab, reguła $b \to a$ mówi zaś, że b ma być zastąpione przez a. Proces przepisywania rozpoczyna się od pewnego ustalonego ciągu początkowego (aksjomatu). Załóżmy, że składa się on początkowo tylko z litery b. Tak zdefiniowany L-system wygląda następująco:

ω: b
p_1: $a \to ab$
p_2: $b \to a$

W pierwszym kroku b zastępowane jest przez a zgodnie z produkcją $b \to a$. W drugim kroku a jest zastąpiony przez ciąg ab, wykorzystując produkcję $a \to ab$. Słowo ab składa się z dwóch liter, do których w następnym kroku *jednocześnie* zastosowane są odpowiednie produkcje. Litera a zastąpiona jest przez ab, zaś b przez a, prowadząc do ciągu aba. W następnym kroku ciąg aba produkuje ciąg $abaab$, który znów produkuje ciąg $abaababa$, ten zaś $abaababaabaab$ itd.

1.1.2 L-systemy stochastyczne

W L-systemach stochastycznych, danemu znakowi może być przyporządkowane więcej niż jedna produkcja, przy czym każda z nich ma określone prawdopodobieństwo, z jaką może być zastosowana. Produkcje mają postać $a \to {}_{(P)}\chi$, przy czym prawdopodobieństwo użycia danej produkcji umieszczone jest tuż za symbolem $\to$. Przykładowy L-system może wyglądać w sposób następujący:

ω: F
p_1: $F \to_{(0.33)} F[+F]F[-F]F$
p_2: $F \to_{(0.33)} F[+F]F$
p_3: $F \to_{(0.34)} F[-F]F$

W przypadku tak zdefiniowanego L-systemu, każda z tych produkcji może być wybrana z prawdopodobieństwem równym w przybliżeniu $1/3$.

1.1.3 L-systemy kontekstowe

W L-systemach kontekstowych produkcja jest zastosowana w stosunku do jakiegoś znaku, wyłącznie wtedy, gdy zgadza się również odpowiedni kontekst (lewy, prawy bądź oba). W *2L-systemach* produkcje są postaci $a_l < a > a_r \to \chi$, co oznacza, że litera a może wyprodukować słowo χ wtedy i tylko wtedy, gdy poprzedza ją a_l zaś po niej następuje a_r. Litery a_l i a_r tworzą w tej produkcji odpowiednio lewy i prawy *kontekst* litery a. Produkcje w *1L-systemach* mają tylko jeden kontekst: lewy lub prawy, są więc postaci $a_l < a \to \chi$ lub $a > a_r \to \chi$.

Przykładowy kontekstowy L-system może wyglądać tak:

ω: $baaaaaaaa$
p_1: $b < a \to b$
p_2: $b \to a$

Kilka początkowych słów przez niego wygenerowanych wygląda następująco:

baaaaaaaa
abaaaaaaa
aabaaaaaa
aaabaaaaa
aaaabaaaa
...

1.1.4 L-systemy parametryczne

W L-systemach parametrycznych z symbolami skojarzone są pewne (jeden lub więcej) parametry. Ponadto produkcje narzucają dodatkowe warunki na te parametry i stosowane są tylko wtedy, gdy zgadza się liczba parametrów i określony warunek jest spełniony. Przykładowa produkcja może wyglądać tak: $A(t) : t > 5 \rightarrow B(t+1)CD(t*0.5, t-2)$. Warunek umieszczony jest pomiędzy znakami ':' a '$\rightarrow$'. Produkcja ta oznacza, że symbol A posiadający jeden parametr, zamieniany jest na ciąg $B(t+1)CD(t*0.5, t-2)$ tylko wtedy, gdy wartość tego parametru jest większa od 5. Oczywiście jako parametry symboli w nowym ciągu wpisywane są odpowiednie wyliczone wartości. Przykładowo symbol $A(6)$ zostanie zastąpiony ciągiem $B(7)CD(3,1)$, natomiast w stosunku do symbolu $A(5)$ produkcja ta nie zostanie zastosowana. Jeśli do symbolu nie zostanie zastosowana żadna produkcja (żaden warunek nie jest spełniony) stosowana jest wtedy domyślna produkcja $a \rightarrow a$.

Przykładowy L-system parametryczny:

ω: $B(2)A(4,4)$
p_1: $A(x,y) : y <= 3 \rightarrow A(x*2, x+y)$
p_2: $A(x,y) : y > 3 \rightarrow B(x)A(x/y, 0)$
p_3: $B(x) : x < 1 \rightarrow C$
p_4: $B(x) : x >= 1 \rightarrow B(x-1)$

oraz kilka początkowych wyrazów przez niego generowanych:

B(2)A(4,4)
B(1)B(4)A(1,0)
B(0)B(3)A(2,1)
CB(2)A(4,3)
CB(1)A(1.33,7)
CB(0)B(1.33)A(0.19,0)

1.2 Interpretacja ciągu znaków

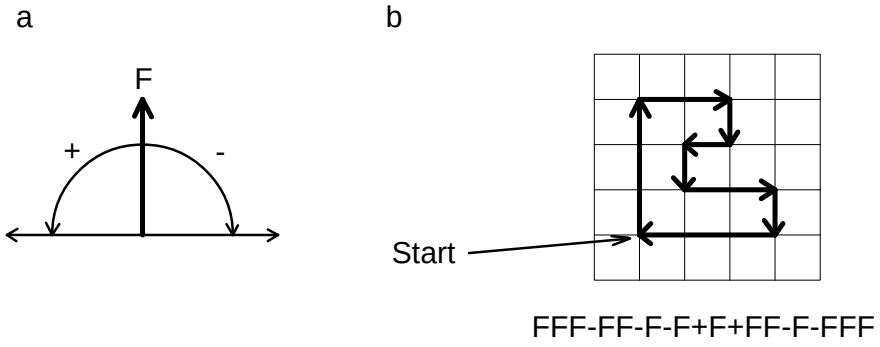
Same ciągi znaków nie są jednak zbyt iteresujące, nawet mimo stosowania tak wyrafinowanych metod ich tworzenia. Najpopularniejszą chyba obecnie metodą interpretacji ciągów znaków jest sterowanie żółwiem (podobnym do żółwia w języku LOGO).

1.2.1 Żółw na płaszczyźnie

W przypadku dwuwymiarowym *stan* żółwia jest opisany trójką (x,y,α), gdzie współrzędne kartezjańskie (x,y) określają aktualną *pozycję* żółwia, zaś kąt α określa kierunek, w który żółw jest zwrócony. Mając dane: długość kroku d oraz przyrost kąta δ żółw może reagować na komendy reprezentowane przez następujące symbole (rysunek 1):

`F` Krok do przodu o długości d. Stan żółwia zmienia się na (x',y',α), gdzie $x' = x + d\cos\alpha$ i $y' = y + d\sin\alpha$. Między punktami (x,y) a (x',y') rysowana jest linia.
`f` Krok do przodu o długości d bez rysowania linii.
`+` Skręt w lewo o kąt δ. Stan żółwia zmienia się w $(x,y,\alpha+\delta)$.
`-` Skręt w prawo o kąt δ. Stan żółwia zmienia się w $(x,y,\alpha-\delta)$.

Mając wejściowy ciąg ν, początkowy stan żółwia (x_0,y_0,z_0), oraz ustalone parametry d i δ, *interpretacją żółwia* ciągu ν jest rysunek (zbiór linii) powstały w wyniku reakcji na komendy zawarte w ciągu ν. W szczególności metoda ta może zostać wykorzystana do interpretacji ciągów wygenerowanych przez L-systemy.

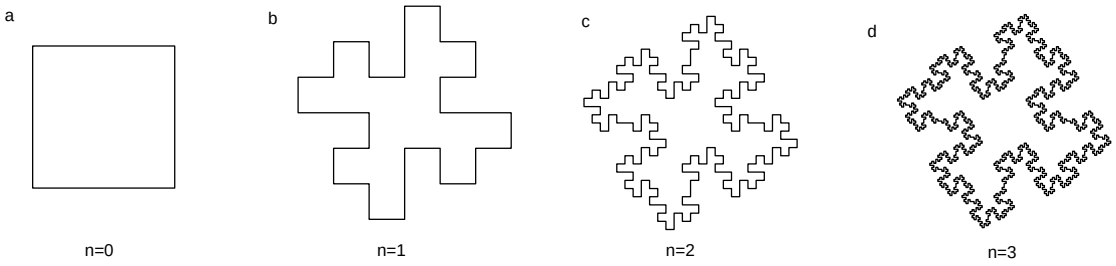


Rysunek 1: (a) Interpretacja symboli F, $-$, $+$. (b) Interpretacja ciągu znaków. Kąt skrętu δ równy $90°$. Początkowo żółw skierowany jest w górę.

Przykładowo rysunek 2 przedstawia rysunki powstałe w wyniku interpretacji ciągów znaków wygenerowanych przez poniższy L-system:

ω: $F+F+F+F$
p_1: $F \rightarrow F-F+F+FF-F-F+F$

Rysunki odpowiadają interpretacji ciągów powstałych w kolejnych krokach (0..3) działania L-systemu.



Rysunek 2: Generowanie kwadratowej wyspy Kocha

1.2.2 Żółw w przestrzeni

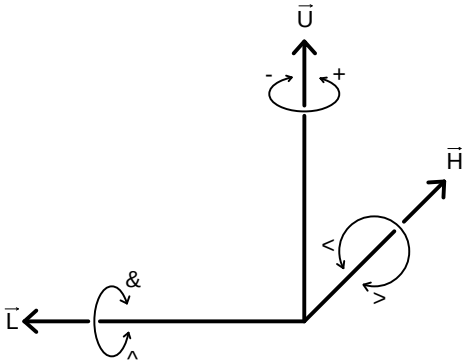
Interpretację żółwia L-systemów można łatwo rozszerzyć do trzech wymiarów. Kluczowym założeniem jest reprezentacja *orientacji* żółwia w przestrzeni za pomocą trzech wektorów $\vec{H},\vec{L},\vec{U}$, reprezentujących odpowiednio kierunki *w przód*, *w lewo* i *w górę*. Wektory te mają długość jednostkową oraz są wzajemnie prostopadłe, tzn. spełniają równanie $\vec{H} = \vec{L} \times \vec{U}$. Obroty żółwia można wykonywać tworząc odpowiednią macierz rotacji „obracającą" wokół pewnego wektora. Przykładowo skręty „w lewo" i „w prawo" odbędą się przez obrócenie wektorów $\vec{H}$ i $\vec{L}$ wokół osi, którą będzie wektor $\vec{U}$.

Orientacja żółwia w przestrzeni kontrolowana jest następującymi symbolami (rysunek 3):

- `+` Skręt w lewo o kąt δ (obrót wokół wektora $\vec{U}$).
- `-` Skręt w prawo o kąt δ (obrót wokół wektora $\vec{U}$).
- `&` Nachylenie w dół o kąt δ (obrót wokół wektora $\vec{L}$).
- `^` Nachylenie w górę o kąt δ (obrót wokół wektora $\vec{L}$).
- `<` Przechył w lewo o kąt δ (obrót wokół wektora $\vec{H}$).
- `>` Przechył w prawo o kąt δ (obrót wokół wektora $\vec{H}$).
- `|` Zawrócenie (obrót o $180°$ wokół wektora $\vec{U}$).

1.2.3 Struktury drzewiaste

Zgodnie z przedstawionymi do tej pory regułami żółw interpretuje ciąg znaków jako sekwencje segmentów linii. W zależności od długości jednego segmentu d i kąta skrętu δ wynikowa krzywa może się ze sobą przecinać



Rysunek 3: Kontrolowanie orientacji żółwia w trzech wymiarach.

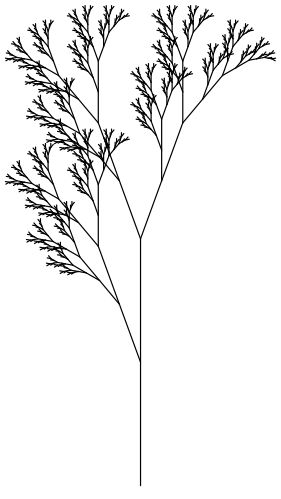
lub nie, może być bardziej lub mniej poplątana, może mieć niektóre segmenty narysowane wielokrotnie, inne zaś mogą być niewidoczne, zawsze jednak jest to pojedyncza krzywa. Jednakże królestwo roślin zdominowane jest przez struktury rozgałęziające się, potrzebna jest więc metoda tworzenia struktur drzewiastych. Jednym z możliwych rozwiązań jest stosowanie ciągów z nawiasami. Poniżej opisano rozszerzenie interpretacji żółwia o ciągi z nawiasami oraz działanie nawiasowych L-systemów.

Do rozgraniczania gałęzi wprowadzone są dwa nowe symbole. Są one interpretowane przez żółwia w sposób następujący:

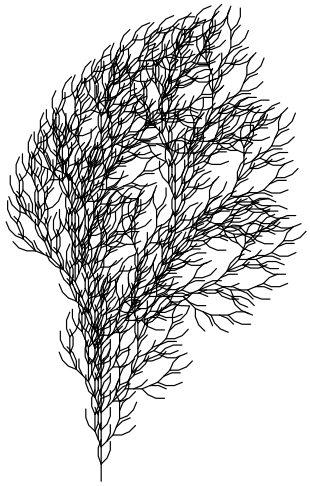
[Zapisz aktualny stan żółwia na stosie. Informacją zapisaną na stosie jest położenie i orientacja żółwia, oraz inne możliwe parametry, takie jak na przykład kolor lub grubość aktualnie rysowanych linii.

] Zdejmij ze stosu stan żółwia. Stan zdjęty ze stosu staje się stanem aktualnym. Nie rysowana jest żadna linia, choć pozycja żółwia się zmienia.

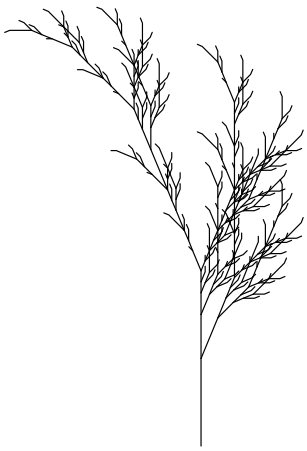
Przykładowe struktury drzewiaste wykonane tą właśnie metodą pokazane są na rysunku 4.



n=7, δ =20
X
X=F[+X]F[-X]+X
F=FF



n=4, δ =22.5
F
F=FF-[-F+F+F]+[+F-F-F]



n=5, δ =22.5
X
X=F-[[X]+X]+F[+FX]-X
F=FF

Rysunek 4: Przykładowe struktury drzewiaste wraz z generującymi je nawiasowymi L-systemami

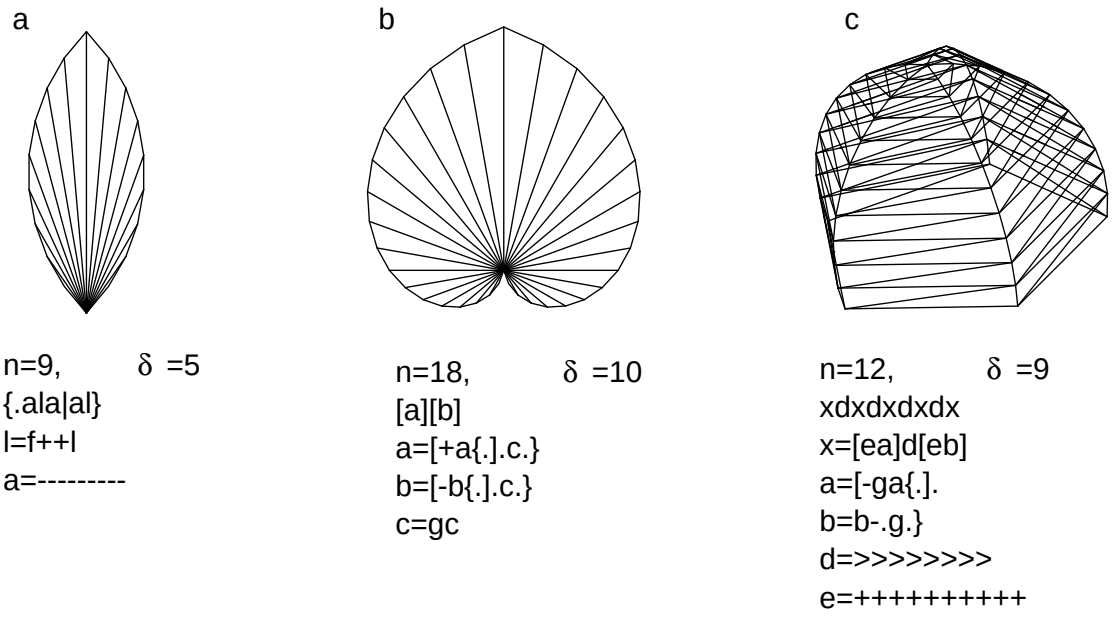
1.2.4 Liście

Do modelowania liści konieczne jest rozszerzenie alfabetu symboli interpretowanych przez żółwia. W tym celu stosowane są symbole { i }. Symbol { oznacza rozpoczęcie tworzenia wielokąta, zaś symbol } zakończenie. W trybie tworzenia wielokąta, każdy ruch za pomącą komend `f` lub `F` powoduje zapisanie nowego wierzchołka do wielokąta. Ruch przy pomocy komendy `g` nie powoduje zapisania wierzchołka, zaś komenda `.` powoduje zapisanie do wielokąta aktualnej pozycji żółwia jako wierzchołka. Tym sposobem można tworzyć figury przypominające liście. Rysunek 5a przedstawia liść wygenerowany przy pomocy obrysowania konturu liścia, tzn. żółw poruszał się po krawędzi liścia tworząc wierzchołki. Natomiast liść z rysunku 5b został stworzony na bazie struktury drzewiastej, tzn. żółw dojeżdżał wielokrotnie z punktu startowego do krawędzi tworząc nowe wierzchołki.

Aby umożliwić tworzenie zagnieżdżeń komend { i } ostatecznie ich interpretacja przez żółwia zdefiniowana jest następująco:

- `{` Rozpoczęcie tworzenia nowego wielokąta: aktualny wielokąt zostaje zapisany na stos wielokątów, następnie tworzony jest nowy (pusty) aktualny wielokąt.
- `.` Dodanie wierzchołka do aktualnego wielokąta.
- `}` Aktualny wielokąt zostaje narysowany na podstawie należących do niego wierzchołków, następnie ze stosu ściągany jest wielokąt, który staje się aktualnym wielokątem.

Przykład wykorzystania zagnieżdżonych wielokątów pokazany jest na rysunku 5c.



Rysunek 5: (a) Liść wygenerowany przy pomocy obrysowania konturu. (b) Liść wygenerowany na bazie struktury drzewiastej. (c) Przestrzenna struktura stworzona za pomocą zagnieżdżonych wielokątów.

2 Implementacja

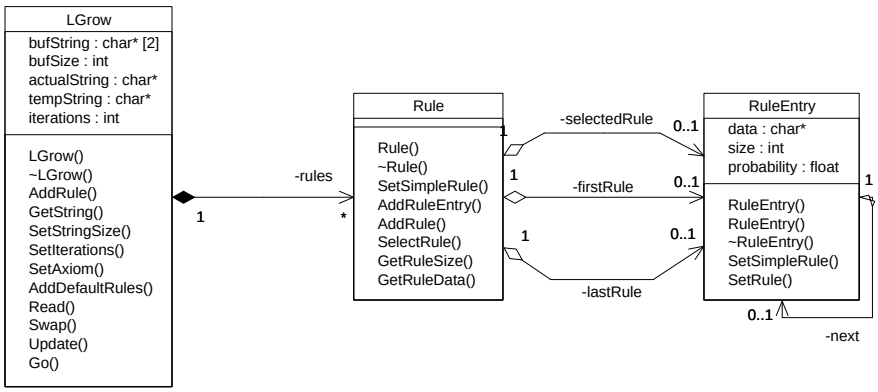
W projekcie zostały zaimplementowane L-systemy bezkontekstowe, stochastyczne. Przykładowa produkcja stochastyczna wygląda następująco: `a=(0.5)ba`. Z cech L-systemów parametrycznych pozostała jedynie możliwość podawania bezpośrednich wartości liczbowych jako parametrów komend sterujących ruchem żółwia (przykładowo `-(30)` oznacza skręt w prawo o 30°).

Budowanie figury geometrycznej za pomocą L-systemów składa się z trzech etapów:

1. Rozwinięcie wejściowego L-systemu w ciąg znaków.
2. Interpretacja ciągu znaków, tj. stworzenie danych o położeniu segmentów i liści w przestrzeni, ich kolorze itp. Dane te mogą być wyeksportowane np. jako dane dla programu POV-Ray (segmenty stworzone przy pomocy komendy `F` są zapisywane jako cylindry, liście jako zbiór trójkątów).
3. Opcjonalna *triangulacja* danych, tzn. stworzenie opisu wszystkich elementów obiektu przy pomocy trójkątów.

2.1 Klasy i funkcje

Klasą odpowiedzialną za *rozrośnięcie* się L-systemu w jeden ciąg znaków jest klasa **LGrow**. Wczytuje ona dane wejściowego L-systemu, tj. ciąg początkowy, produkcje oraz ilość iteracji. Następnie ciąg początkowy jest *rozwijany* tj. stosowane są odpowiednie produkcje. Każdy znak ciągu zostaje zastąpiony ciągiem znaków z odpowiedniej produkcji skojarzonej z danym znakiem. Wyjątkiem od tej reguły są znaki znajdujące się pomiędzy `(` a `)`, do nich zawsze stosowane są produkcje $a \to a$. Reguły przechowywane są w tablicy składającej się z elementów **Rule**, z których każdy przechowuje listę elementów **RuleEntry**, tj. ciągów, które mają dany znak zastąpić, przy czym każdy z nich ma ustalone odpowiednie prawdopodobieństwo. Jeśli dana produkcja nie ma określonego prawdopodobieństwa (zawsze wykonywana), w elemencie **Rule** istnieje wtedy tylko jeden element **RuleEntry** z prawdopodobieństwem równym 1.

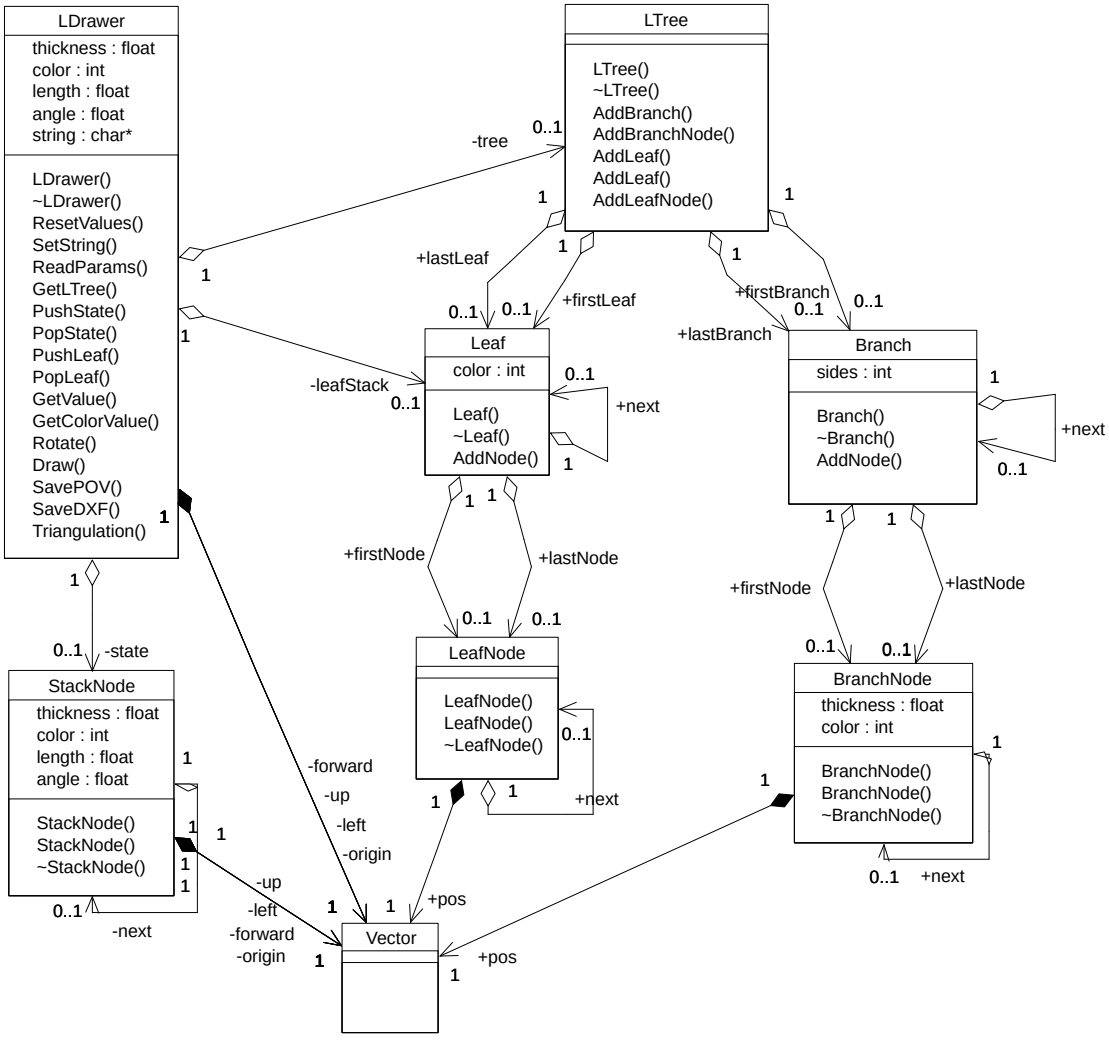


Rysunek 6: Powiązania klas odpowiedzialnych za rozrost L-systemu.

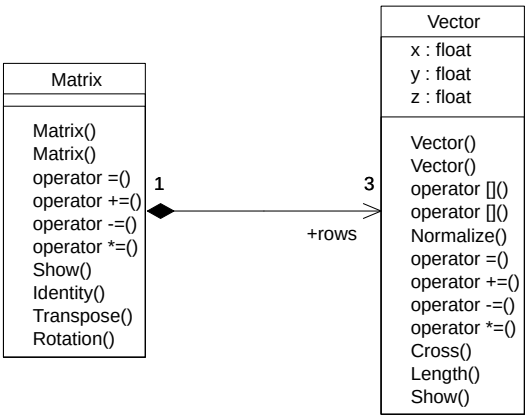
Wyjściowy ciąg znaków jest interpretowany przez klasę **LDrawer**. Tworzy ona przestrzenną strukturę **LTree**, składającą się z listy gałęzi **Branch**, z których każda składa się z listy węzłów **BranchNode**. W każdym z węzłów przechowywane są informacje o: położeniu węzła w przestrzeni, grubości węzła, kolorze węzła. Ponadto w strukturze **LTree** znajduje się lista liści **Leaf**, gdzie każdy z liści posiada informację o kolorze oraz listę wierzchołków liścia **LeafNode** zawierających informacje o położeniu w przestrzeni. Dane zawarte w strukturze **LTree** mogą być bezpośrednio zapisane do pliku programu POV-Ray.

Klasa **LTriangulator** zajmuje się triangulacją danych ze struktury **LTree**.

Ponadto wykorzystywane są pomocnicze klasy: **Vector** do operacji na wektorach trójwymiarowych, oraz **Matrix** do operacji na macierzach 3×3. (Ułatwiające przede wszystkim poruszanie żółwiem przy interpretacji ciągu znaków).



Rysunek 7: Powiązania klas odpowiedzialnych za interpretację ciągów znaków.



Rysunek 8: Pomocnicze klasy do operacji na wektorach i macierzach

2.2 Dostępne komendy

2.2.1 Komendy obracające

`+`, `+(`n`)` Skręt w lewo
`-`, `-(`n`)` Skręt w prawo
`&`, `&(`n`)` Nachylenie w dół
`^`, `^(`n`)` Nachylenie w górę
`<`, `<(`n`)` Przechył w lewo
`>`, `>(`n`)` Przechył w prawo

Podawanym opcjonalnie parametrem jest kąt w stopniach. Jeśli parametr nie jest podany, używana jest domyślna wartość kąta.

2.2.2 Specjalne komendy obracające

`|` Zawrócenie (obrót o 180°)
`~`, `~(`n`)` Obrót wokół każdej z osi o kąt, którego wartość jest liczbą losową z przedziału $[-n, n]$ (Domyślnie $n = 10$)
`$` Obrót wzdłuż osi $\vec{H}$ o taki kąt, aby wektor $\vec{L}$ (kierunek *w lewo*) znalazł się w pozycji poziomej (równoległej do podłoża)
`t`, `t(`n`)` Nachylenie w dół (symulowanie grawitacji)

2.2.3 Komendy przesuwające żółwia

`F`, `F(`n`)` Przesunięcie do przodu, narysowanie segmentu, utworzenie wierzchołka liścia (jeśli włączony tryb tworzenia liści)
`f`, `f(`n`)` Przesunięcie do przodu, utworzenie wierzchołka liścia (jeśli włączony tryb tworzenia liści)
`g`, `g(`n`)` Przesunięcie do przodu.

Przesunięcie o domyślną długość segmentu, chyba że podano parametr.

2.2.4 Komendy tworzące rozgałęzienia

`[` Stworzenie rozgałęzienia (zapis stanu żółwia na stos)
`]` Powrót z rozgałęzienia (odczytanie stanu żółwia ze stosu)

2.2.5 Komendy tworzące liście (wielokąty)

`{` Utworzenie na stosie wielokątów nowego pustego wielokąta (liścia)
`.` Wpisanie do aktualnego liścia aktualnej pozycji żółwia
`}` Zdjęcie liścia ze stosu wielokątów i narysowanie go.

2.2.6 Komendy modyfikujące domyślne wartości parametrów

`"`, `"(`n`)` Zwiększenie domyślnej długości segmentu ($\times 1.1$)
`'`, `'(`n`)` Zmniejszenie domyślnej długości segmentu ($\times 0.9$)
`;`, `;(`n`)` Zwiększenie domyślnej wartości kąta skrętu ($\times 1.1$)
`:`, `:(`n`)` Zmniejszenie domyślnej wartości kąta skrętu ($\times 0.9$)
`?`, `?(`n`)` Zwiększenie grubości segmentu ($\times 1.4$)
`!`, `!(`n`)` Zmniejszenie grubości segmentu ($\times 0.7$)

Jeśli brak parametru komendy, wartości zmieniają się o podane stałe.

2.2.7 Inne komendy

`c(#`*rrggbb*`)` Ustawienie koloru. Kolor podawany jest podobnie jak w HTML: szesnastkowe wartości każdej ze składowych, całość poprzedzona znakiem #.

2.3 Format pliku wejściowego

Przykładowy plik wejściowy wygląda następująco:

```
# ilość iteracji
@i=10
# domyślny kąt skrętu
@a=20
# ciąg początkowy
=A
# reguły rozrostu
A=A[+FB][-FC]
B=F^B
C=F&C
@@
```

- Linie rozpoczynające się znakiem '#' są ignorowane, umożliwiając pisanie komentarzy.
- Linie rozpoczynające się znakiem '@' służą do ustalenia parametrów L-systemu:

  `@i=`*n* - ilość iteracji (*iterations*)

  `@a=`*n* - kąt skrętu (*angle*)

  `@l=`*n* - długość segmentu (*length*)

  `@t=`*n* - szerokość segmentu (*thickness*)

  ponadto linia rozpoczynająca się ciągiem '@@' oznacza koniec danych (dalsza część pliku, o ile istnieje, jest ignorowana).
- Linia rozpoczynająca się znakiem '=' określa ciąg początkowy (aksjomat) - jest nim ciąg znaków do końca linii, rozpoczynający się za znakiem '='.
- Linie, w której drugim znakiem jest '=' traktowane są jako reguły rozrostu (produkcje): znak znajdujący się na początku linii ma być zastępowany ciągiem znaków znajdujących się za znakiem '='.

2.4 Program

Program wymaga podania jako parametru nazwy pliku z rozszerzeniem `.ls`, zawierającego specyfikację L-systemu. Opcjonalnie można podać nazwę wynikowego pliku, do którego zapisywane są obiekty dla POV-Ray'a, w przypadku braku nazwy pliku wynikowego nazwa ta tworzona jest z nazwy pliku wejściowego. Opcjonalny parametr `-s` powoduje wypisanie na ekran ciągu znaków wyprodukowanego przez L-system.

Bibliografia

[1] Przemysław Prusinkiewicz and Aristid Lindenmayer;
The Algorithmic Beauty of Plants, Springer-Verlag, New York, 1990.

[2] Przemysław Prusinkiewicz, Mark Hammel, and Radomir Mech;
Visual Models of Morphogenesis: A Guided Tour, 1997.
`http://www.cpsc.ucalgary.ca/projects/bmv/vmm/title.html`

[3] Laurens Lapre;
Lparser.
`http://www.xs4all.nl/~ljlapre/lparser.htm`

[4] David J. Wright;
Dynamical Systems and Fractals, 1996.
`http://www.math.okstate.edu/mathdept/dynamics/lecnotes/lecnotes.html`

[5] David G. Green;
L-System tutorial.
`http://life.csu.edu.au/complex/tutorials/tutorial2.html`

[6] Hung-Wen Chen;
L-System Plant Geometry Generator, 1995.
`http://www.tc.cornell.edu/Visualization/contrib/cs490-94to95/hwchen/index.html`