



UMCS

UNIwersytet Marii Curie-Skłodowskiej
w Lublinie

Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Specjalność: –

Aleh Hutsko

Nr albumu: 296609

Symulacja wzrostu roślin generowanych przez system Lindenmayera

Simulation of the growth of plants generated by the
Lindenmayer system

Praca licencjacka

napisana w Katedrze oprogramowania systemów informatycznych
pod kierunkiem dr Krzysztofa Dmitruka

LUBLIN 2022

Spis treści

Wstęp	5
1 System Lindenmayera	7
1.1 Informacje wstępne	7
1.2 Struktura L systemu	7
1.3 Interpretacja ciągu znaków	10
2 Implementacja	15
2.1 Wykorzystane narzędzia	15
2.1.1 Język i środowisko	15
2.1.2 Biblioteka Proctree	15
2.1.3 Biblioteka nlohmann Json	15
2.1.4 OpenGL	15
2.2 Funkcjonalność aplikacji	15
2.2.1 Planowanie drzew	15
2.2.2 Symulacja wzrostu	15
2.2.3 Ustawianie parametrów	15
2.2.4 Ustawianie tekstur	15
2.2.5 Zapis do pliku	15
2.3 Struktura programu	15
3 Testy i rezultaty	17
3.1 Wydajność	17
3.2 Porównanie z innymi rozwiązaniami	17
4 Podsumowanie	19

Wstęp

Rośliny to rozległa grupa organizmów żywych, występujących na większości lądów na Ziemi, a także w środowisku wodnym. Należą do nich trawy, drzewa, kwiaty, krzewy, paprocie, mchy i wiele innych. Istnieje około 391,000 gatunków roślin, z których zdecydowana większość, około 369,000 (94%), wytwarza nasiona.[1] Rośliny można znaleźć na całym świecie, na wszystkich kontynentach. Rośliny dostarczają znaczną część tlenu na świecie i stanowią podstawę większości ekosystemów na Ziemi. Tak ważna część świata rzeczywistego prędzej czy później wymagała matematycznego opisu i dalszego zastosowania w różnych rodzajach nauki, w szczególności w informatyce. Modelowanie roślin w informatyce jest szeroko stosowane w wielu dziedzinach, takich jak gry, przemysł filmowy, agrokultura i architektura. Rośliny charakteryzują się złożoną, zwykle fraktalną strukturą, która jest trudna do modelowania. Z tego powodu opracowano różne systemy opisywania modeli roślin, aby uporządkować i uprościć pracę z modelowaniem drzew. Jednym z takich systemów jest system Lindenmaiera, który umożliwia opis struktur fraktalnych, w szczególności roślin na poziomie gramatyki formalnej.

Celem tej pracy jest analiza i zapoznanie się z systemem Lindenmayera, możliwościami jego rozbudowy i wykorzystania do generowania roślin, a konkretnie drzew. Ponadto należy opracować oprogramowanie umożliwiające tworzenie trójwymiarowych modeli drzew z możliwością modyfikacji parametrów drzew i symulacji ich wzrostu. Oprogramowanie powinno posiadać następujące funkcje:

- możliwość wyświetlania drzew w przestrzeni trójwymiarowej,
- możliwość modyfikowania drzew przy użyciu różnych parametrów,
- możliwość wyboru tekstur dla pnia drzewa i liści,
- możliwość symulacji wzrostu drzew,

-
- możliwość zapisywania i wczytywania drzew o określonych parametrach.

Rozdział 1

System Lindenmayera

1.1 Informacje wstępne

Systemy Lindenmayera (L-Systemy) zostały wprowadzone i rozwinięte w 1968 roku przez Aristida Lindenmayera, węgierskiego biologa teoretycznego i botanika z Uniwersytetu w Utrechcie. Lindenmayer wykorzystał L-systemy do opisu zachowania komórek roślinnych i modelowania procesów wzrostu w rozwoju roślin.

Reguły L-systemu reprezentują rekurencję. Prowadzi to do samopodobieństwa, a więc formy fraktalne można łatwo opisać za pomocą L-Systemu. Modele roślin, komórek i innych form organicznych naturalnie występujących gatunków można łatwo zdefiniować za pomocą L-systemu, ponieważ wraz ze wzrostem poziomu rekurencji forma powoli "rośnie" i staje się coraz bardziej złożona. Systemy Lindenmayera są również popularne w symulowaniu sztucznego życia.

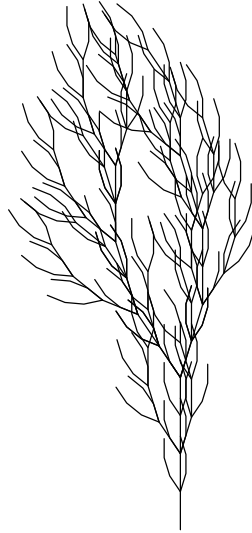
Na rysunku 1.1 jest przykład zastosowania L-systemu dla stworzenia fraktalnej struktury, która przypomina drzewo.

1.2 Struktura L systemu

L-systemy są obecnie powszechnie nazywane parametrycznymi L-systemami, definiowanymi jako krotka:

$$G = (V, \omega, P), \tag{1.1}$$

gdzie



Rysunek 1.1: Przykład stworzonej struktury za pomocą L-systemu

- V (alfabet) – to zbiór symboli zawierający zarówno elementy, które można zastąpić (zmiennie), jak i te, których nie można zastąpić ("stałe" lub "terminale")
- ω (początek, aksjomat lub inicjator) – to ciąg symboli z V , który określa stan początkowy systemu.
- P – to zbiór reguł produkcji lub produktów określających sposób zastępowania zmiennych przez kombinacje stałych i innych zmiennych. Produkcja składa się z dwóch ciągów: poprzednika i następnika. Dla każdego symbolu A , który jest członkiem zbioru V i nie występuje po lewej stronie żadnego iloczynu w P , zakłada się tożsamość iloczynu $A \rightarrow A$; symbole te nazywamy stałymi lub terminalnymi.

W standardowej wersji L-systemów reguły wnioskowania są następujące:

$$v \rightarrow \omega, \quad (1.2)$$

gdzie v jest znakiem danego alfabetu V , $\omega \in V^*$ jest łańcuchem znaki (ewentualnie puste) w tym samym alfabecie. Każdą regułę można więc interpretować jako podział komórki ($|\omega| > 1$), lub jej modyfikację ($|\omega| = 1$), lub jako jej śmierć ($|\omega| = 0$).

Na tabeli 1.1 przedstawiono przykład L-systemu.

Po zdefiniowaniu L-systemu, zaczyna ona ewoluować zgodnie ze swoimi zasadami. Stanem początkowym L-systemu jest jego aksjomat. Wraz z dalszym rozwojem ta linia opisująca stan ulegnie zmianie. Rozwój L-systemu odbywa się cyklicznie. W każdym cyklu rozwoju ciąg jest oglądany od początku do końca, symbol po symbolu. Dla każdego

Tabela 1.1: Przykład stworzonej struktury za pomocą L-systemu

Alfabet	Aksjomat	Reguły
$\{ \ulcorner A \urcorner, \ulcorner B \urcorner, \ulcorner F \urcorner, \ulcorner H \urcorner, \ulcorner J \urcorner, \ulcorner + \urcorner, \ulcorner - \urcorner \}$	$\ulcorner FB \urcorner$	$\ulcorner A \urcorner \rightarrow \ulcorner FBFA + HFA + FB - FA \urcorner$ $\ulcorner B \urcorner \rightarrow \ulcorner FB + FA - FB - JFBFA \urcorner$ $\ulcorner F \urcorner \rightarrow \ulcorner \urcorner$ $\ulcorner H \urcorner \rightarrow \ulcorner - \urcorner$ $\ulcorner J \urcorner \rightarrow \ulcorner + \urcorner$

znaku wyszukiwana jest reguła, dla której ten znak jest poprzednikiem. Jeśli taka reguła nie zostanie znaleziona, znak jest pozostawiony bez zmian. Innymi słowy, dla tych znaków $\ulcorner X \urcorner$, dla których nie istnieje reguła jawna, obowiązuje reguła domyślna: $\ulcorner X \urcorner \rightarrow \ulcorner X \urcorner$. Jeśli zostanie znaleziona pasująca reguła, znak poprzednika jest zastępowany przez łańcuch następnika z tej reguły.

Dla ilustracji rozważmy następujący L-system (nazywamy go glon (łat. *Algæ*), ponieważ jego rozwój modeluje wzrost pewnego gatunku alg) w tabeli 1.2:

Tabela 1.2: Przykład stworzonej struktury za pomocą L-systemu

Aksjomat	Reguły
$\ulcorner A \urcorner$	$\ulcorner A \urcorner \rightarrow \ulcorner B \urcorner$ $\ulcorner B \urcorner \rightarrow \ulcorner AB \urcorner$

W tabeli 1.3 przedstawiono stany tego L-systemu odpowiadające pierwszym dziesięciu cyklom rozwoju systemu.

Można zauważyć, że długości ciągów kodujących stan takiego L-systemu tworzą ciąg liczb Fibonacciego, czyli ciąg liczbowy, w którym każda liczba jest równa sumie dwóch poprzednich. Ciągami Fibonacciego będą także numery znaków A i B w tych ciągach. Bardziej zaskakujący jest fakt, że ciąg ciągów ma taką samą prawidłowość jak ciąg liczb Fibonacciego: każdy ciąg jest sumą (konkatenacją) dwóch poprzednich.

Aby uzyskać stan l-systemu po określonej liczbie iteracji, napisałem funkcję (listing 1.1), do której można wstawić aksjomat, zbiór reguł l-systemu oraz liczbę iteracji.

Tabela 1.3: Wyniki l-systemu z tabeli 1.2 od zera do ośmiu iteracji

Generacja	Stan
0	「A」
1	「B」
2	「AB」
3	「BAB」
4	「ABBAB」
5	「BABABBAB」
6	「ABBABBABABBAB」
7	「BABABBABABBABABBAB」
8	「ABBABBABABBABABBABABBABABBABABBAB」

Funkcja zwraca stan łańcucha po podanej liczbie iteracji.

```

1  def iter(axiom: str, rules: dict, iterations: int) -> str:
2      if iterations == 0: return axiom
3      returnString = ''
4      for i in axiom:
5          if i in rules:
6              returnString += rules[i]
7          else:
8              returnString += i
9      return iter(returnString, rules, iterations-1)

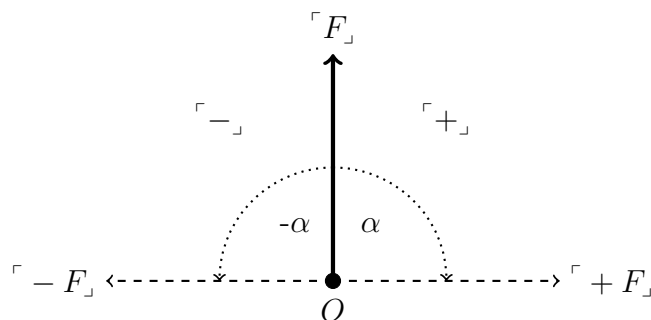
```

Listing 1.1: Funkcja, która zwraca stan systemu po określonej liczbie iteracji

1.3 Interpretacja ciągu znaków

W celu dalszej graficznej interpretacji otrzymanych ciągów należy wprowadzić pojęcie grafiki żółwia. Grafika żółwia to zasada organizacji graficznej biblioteki wyjściowej oparta na metaforze żółwia, wyimaginowanego (a w niektórych eksperymentach rzeczywistego) urządzenia przypominającego robota, które porusza się po ekranie lub papierze i obraca w zadanym kierunku, pozostawiając za sobą (lub opcjonalnie nie pozostawiając) narysowaną linię o zadanym kolorze i grubości.

Interpretacja znaków polega na zdefiniowaniu operacji dla symboli (nie jest konieczne dla wszystkich) w alfabecie. Czynności, podobnie jak symbole, są z kolei definiowane przez autora systemu. Rysunek 1.2 przedstawia przykład interpretacji symbolu (z kątem $\alpha = 90^\circ$) w następujący sposób:



Rysunek 1.2: Przykładowa interpretacja symboli

- $\lceil F \rceil$ oznacza przejście do przodu i narysuj linię
- $\lceil - \rceil$ oznacza obrót w kierunku przeciwnym do ruchu wskazówek zegara na kąt o mierze α
- $\lceil + \rceil$ oznacza obrót zgodnie z ruchem wskazówek zegara o α

Zdefiniujemy również zbiór reguł L-systemu w tabeli 1.4. Łącząc wyżej wymieniony zestaw reguł z interpretacją symboli z rysunku 1.2, otrzymujemy strukturę rekurencyjną zwaną krzywą smoka (tabela 1.5).

Tabela 1.4: Zestaw reguł L-system dla krzywej smoka (ang. *dragon curve*)

Alfabet	Aksjomat	Reguły
$\{ \lceil F \rceil, \lceil X \rceil, \lceil Y \rceil \}$	$\lceil FB \rceil$	$\lceil A \rceil \rightarrow \lceil FBFA + HFA + FB - FA \rceil$ $\lceil B \rceil \rightarrow \lceil FB + FA - FB - JFBFA \rceil$ $\lceil F \rceil \rightarrow \lceil$ $\lceil H \rceil \rightarrow \lceil - \rceil$ $\lceil J \rceil \rightarrow \lceil + \rceil$

Tabela 1.5: krzywa smoka w iteracji 1-7

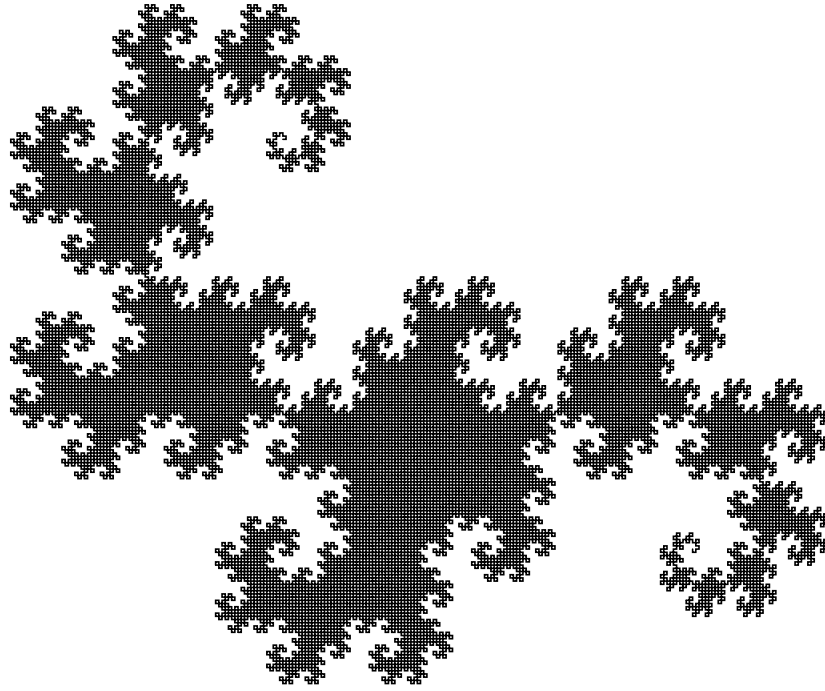
Iteracji	Stan
1	┐
2	┐└
3	┐└┐└
4	┐└┐└┐└┐└
5	┐└┐└┐└┐└┐└┐└
6	┐└┐└┐└┐└┐└┐└┐└┐└┐└
7	┐└┐└┐└┐└┐└┐└┐└┐└┐└┐└┐└┐└

Widać, że po każdej iteracji struktura staje się coraz bardziej złożona. Wynik piętnastu iteracji pokazano na rysunku 1.3.

W przykładzie krzywej smoka (rysunek 1.3) żółw zawsze porusza się na tę samą odległość, a jego linie mają tę samą grubość. Jednak w świecie rzeczywistym rośliny i drzewa mają zasadniczo strukturę rozgałęzioną. W strukturze fraktalnej roślin każda pojedyncza gałąź może być przedstawiona jako osobna roślina, choć w zredukowanej formie. Dlatego, aby symulować drzewa, do l-systemu należy dodać parametry długości i grubości linii utworzonej przez żółwia.

Kolejny zestaw reguł (tabela 1.6) zademonstruje przykład tworzenia struktury przypominającej drzewo. Przykład zaczerpnięty z książki “Alogirytmiczne piękno roślin” [2] (*ang. “Alogirithmic Beauty of Plants”*), której współautorem jest sam Aristid Lindenmayer. W przykładzie tego l-systemu do alfabetu wprowadzane są nowe znaki. Ich interpretacja przez żółwia jest następująca:

- \lceil oznacza “zapisać bieżący stan żółwia do stosu”,



Rysunek 1.3: Wynik piętnastu iteracji

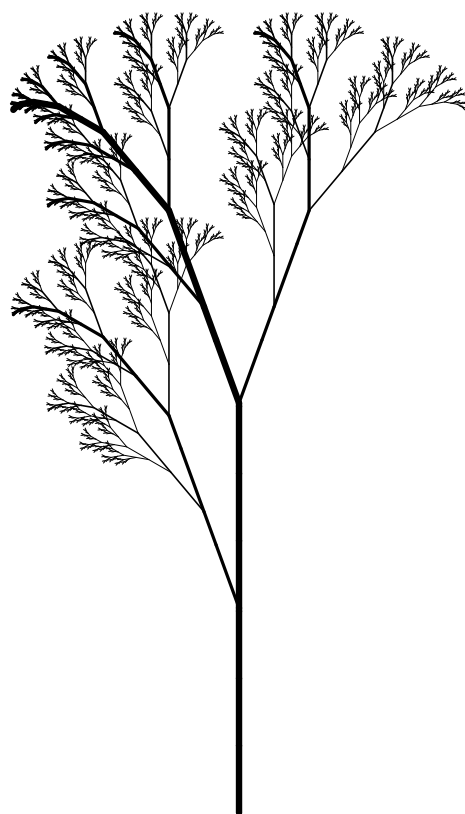
- $\lceil \rfloor$ oznacza “odczytać bieżący stan żółwia do stosu”,
- $\lceil X \rfloor$ jest ignorowany przez żółwia

W przykładzie przedstawionym w książce grubość linii narysowanej przez żółwia nie zmienia się. Rozszerzyłem przykład z podręcznika i dodałem czynnik grubości, przez który będzie mnożona grubość w zależności od zagnieżdżenia gałęzi.

W wyniku zastosowania wszystkich powyższych reguł powstaje struktura przypominająca drzewo (rysunek 1.4)

Tabela 1.6: Zestaw reguł L-system dla przykładowego drzewa

Alfabet	Aksjomat	Reguły	Konstanty
$\{ \lceil F \rfloor, \lceil X \rfloor, \lceil [\rfloor, \lceil] \rfloor \}$	$\lceil X \rfloor$	$\lceil X \rfloor \rightarrow \lceil F[+X]F[-X] + X \rfloor$ $\lceil F \rfloor \rightarrow \lceil F \rfloor$	Kąt powrotu = 20° Grubość linii = 6px Czynnik grubości = 0.5



Rysunek 1.4: Utworzona struktura na podstawie reguł z tabeli 1.6

Rozdział 2

Implementacja

2.1 Wykorzystane narzędzia

2.1.1 Język i środowisko

2.1.2 Biblioteka Proctree

2.1.3 Biblioteka nlohmann Json

2.1.4 OpenGL

2.2 Funkcjonalność aplikacji

2.2.1 Planowanie drzew

2.2.2 Symulacja wzrostu

2.2.3 Ustawianie parametrów

2.2.4 Ustawianie tekstur

2.2.5 Zapis do pliku

2.3 Struktura programu

Rozdział 3

Testy i rezultaty

3.1 Wydajność

3.2 Porównanie z innymi rozwiązaniami

Rozdział 4

Podsumowanie

Bibliografia

- [1] C. M. and B. W. J, “The number of known plant species in the world and its annual increase,” *researchgate.net*, 2016. [Online; dostęp 19 czerwca 2022].
- [2] P. P. and L. A., *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.