Akademia Górniczo-Hutnicza
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Wykorzystanie L-systemów do modelowania roślin

Przedmiot:
Modelowanie i symulacja systemów
Prowadzący:
mgr inż. Jakub Porzycki

Autorzy:
Tomasz Borowicz
Krzysztof Świder
Arkadiusz Chmiel
Informatyka, III rok

0. Spis treści

1.Wprowadzenie do tematu

2. Wstęp

Nazwa L-systemów pochodzi od nazwiska ich twórcy Lindenmayera. Przez lata systemy te były obiektem zainteresowań wielu naukowców. Systemy te są wykorzystywane głównie do realistycznego modelowania roślin czy konstruktów gramatycznych [1].

Na początku L-systemy były zdefiniowane jako linearne tablice czy skończone automaty, później jednak zostały przetworzone w bardziej wygodne konstrukty gramatyczno podobne, które umożliwiły symulacje tworzenia realistycznych struktur roślinnych.

Z biologicznego punktu widzenia stanowią świetny model teoretyczny który umożliwia dyskusje i porównywanie wyników natury rozwoju zachowania komórkowego. Zaowocowało to różnymi interesującymi rezultatami w dziedzinie biologii.

Zaś z matematycznego punktu widzenia L-systemy otworzyły całkowicie nowy wymiar w teorii języków formalnych. Ich innowacyjność przejawia się w nowych typach problemów oraz sposobów na ich rozwiązywanie [4].

L systemy są bardzo podobne do gramatyk Chomsky'ego z tą różnicą, że produkcja w L-systemach są aplikowane równolegle, a w gramatykach Chomsky'ego sekwencyjnie. L-systemy definiujemy w następujący sposób $G = \{V, \omega, P\}$, gdzie V to alfabet, ω to słowo startowe, a P to zbiór produkcji. Słowo generujemy aplikując równolegle produkcje do słowa początkowego n razy. Wygenerowane słowo zawsze będzie typu $String$ [2][3].

3. Interpretacja graficzna w 2D

Jeżeli chcemy utworzyć geometryczny obraz systemu musimy zastosować tzw. “Interpretację żółwik” (Turtle interpretation). Nazwa ta pochodzi od popularnego żółwika z programu LOGO, do którego analogie można znaleźć w tym algorytmie. Stan naszego kursora (żółwika) zdefiniujemy poprzez trójkę (x, y, α) ,gdzie (x, y) to aktualna pozycja kursora, a α to odchylenie od kąta startowego (kierunek patrzenia żółwika) [3]. Wygenerowany wyraz czytamy od lewej do prawej interpretując znak po znaku. Każdy znak powinien dać się przetłumaczyć na jedną z 4 komend:

- zrób krok długości d* do przodu i narysuj linię. Stan żółwika zmienia się wtedy w następujący sposób: $(x, y, \alpha) \rightarrow (x + d\cos(\alpha), y + d\cos(\alpha), \alpha)$ **[F]**

- zrób krok długości d do przodu i nie rysuj linii. Stan żółwika zmienia się wtedy w następujący sposób: $(x, y, \alpha) \rightarrow (x + d\cos(\alpha), y + d\cos(\alpha), \alpha)$ **[f]**
- zrób obrót w lewo o kąt δ *. Stan żółwika zmienia się wtedy w następujący sposób: ($(x, y, \alpha) \rightarrow (x, y, \alpha + \delta)$ **[+]**
- zrób obrót w prawo o kąt δ . Stan żółwika zmienia się wtedy w następujący sposób: ($(x, y, \alpha) \rightarrow (x, y, \alpha - \delta)$ **[-]**

Po przetłumaczeniu wszystkich znaków otrzymujemy listę komend na podstawie, której możemy wygenerować geometryczny obraz L-systemu[3].

4. Interpretacja graficzna w 3D

Przedstawioną powyżej interpretację można rozszerzyć do trzech wymiarów. W takim przypadku stan kursora opisany jest przez 2 wektory: $\overline{X}, \overline{K}$. Wektor X opisuje aktualne położenie żółwika w trzech płaszczyznach (x, y, z) , a wektor K określają kierunek kursora (kierunek patrzenia żółwika). W interpretacji 3d rotacje realizuje się poprzez przemnożenie wektora K przez odpowiednia macierz[3]:

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

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

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

W tym przypadku obsługiwane komendy to:

- skręć w lewo/prawo o kąt δ używając macierzy rotacji $R_U(\delta) / R_U(-\delta)$ **[+/-]**
- skręć w górę/dół o kąt δ używając macierzy rotacji $R_L(\delta) / R_L(-\delta)$ **[&/^]**
- obróć się wokół własnej osi w lewo/prawo o kąt δ używając macierzy rotacji $R_H(\delta) / R_H(-\delta)$ **[\ lub /]**
- zrób krok długości d do przodu i rysuj linię. **[F]**

5. Struktury z rozgałęzieniami

Rozpatrywane do tej pory systemy opisują proste struktury bez rozgałęzień. W przyrodzie najczęstsze są jednak obiekty posiadające gałęzie (drzewa, krzewy itp.). Aby w prosty sposób symulować takie obiekty do zestawu naszych komend dodać musimy dwie kolejne[3]:

- zapisz aktualną pozycję na stosie. **[[ ]**
- wczytaj pozycję ze stosu i przejdź do niej **[ ]]**

Komendy te najczęściej zapisuje się w postaci nawiasów (np. () lub [] lub <>). Pozwala to w czytelny sposób wyróżnić rozgałęzienia (obiekt w nawiasie to gałąź, natomiast obiekt poza nawiasem to rdzeń).

6. Losowość w L-systemach

Kolejnym problemem przedstawionych do tej pory systemów może być fakt ,że są one deterministyczne (dla jednego słowa początkowego zawsze dają taki sam efekt). Rozwiązaniem tego skonstruowanie systemu, w którym produkcje nie muszą być deterministyczne, a każda produkcja może być zaaplikowana z danym prawdopodobieństwem.

7. Parametryczne L-systemy

Rozwinięciem naszych L-systemów będzie dodanie funkcji parametrycznych. Potrzeba ich użycia pojawia się gdy chcemy odpowiednio odtworzyć rozrost obiektu. W przyrodzie rzadko zdarza się aby odległości i kąty w poszczególnych rozgałęzieniach były stałe. Prostym rozwiązaniem tego problemu jest stworzenie odpowiedniego alfabetu zawierającego przekształcenia o różnych kątach i długościach kroku. Szybko zdajemy sobie jednak sprawę, że takie podejście prowadzi do nadmiernie rozbudowanych definicji systemu (bardzo długi alfabet, bardzo długie zbiory produkcji itp.). Lepszym rozwiązaniem jest zastosowanie funkcji parametrycznych. Przykład produkcji parametrycznej może wyglądać następująco: $A(t) : \ t > 5 \rightarrow B(t+1)CD(t \wedge 0.5, t+7)$. Podczas ostatniego kroku, czyli interpretacji graficznej, można wtedy przekazać parametr do funkcji (np. zrób t kroków do przodu). Warto zauważyć, że takie produkcje pozwalają zdefiniować warunki logiczne ich wystąpienie. Trzeba pamiętać także, że w tym przypadku znaki “(“ oraz “)” są zarezerwowane dla wyróżniania parametrów wejściowych[3][7].

8. Modelowanie drzew

Aby w realistyczny sposób modelować drzewa należy nałożyć na nasze systemy kilka obostrzeń. Po pierwsze wykluczyć należy przecinanie się gałęzi w naszym obiekcie(w przyrodzie rzadko zdarza się aby gałąź zrastała się z inną gałęzią). Należy także w jak naj realistyczniejszy sposób oddać prawa fizyki mające wpływ na rozwój obiektu. Najprostszy model, który zapewnia takie warunki to model Hondy. Zakłada on:

- segmenty drzewa są proste
- segment matka(rdzeń) produkuje dwa segmenty potomne(gałęzie) w jednym procesie rozgałęzienia
- długość segmentów potomnych to długość segmentu matki skrócone o współczynniki r_1, r_2
- segment matka oraz segmenty potomne agregowane są jako jedna gałąź
- segmenty potomne odchodzą od segmentu matki pod kątami α_1, α_2
- segment potomny jest jest poprawiany tak aby być jak najbliżej kolejnej gałęzi
- wyjątki od powyższych reguł mogą zajść tylko dla rdzenia drzewa.

Model ten dzięki modyfikacji jego parametrów umożliwia skuteczne modelowanie szkieletów drzew [3].

9. Realistyczne modelowanie roślin

Realistyczne modelowanie roślin wymaga uwzględnienia dodatkowych czynników. Po pierwsze dla każdego węzła (najbardziej wysuniętej pozycji kursora w każdej gałęzi) w każdej iteracji rozrostu musimy zdecydować czy:

- węzeł zmieni się w kwiat
- węzeł zmieni się w liść
- węzeł w tej iteracji nie rozrasta się
- węzeł rozrasta się w standardowy sposób
- węzeł umiera

Decyzję tą można uwzględnić poprzez odpowiednie produkcje (pozwala to na zdefiniowanie możliwych kształtów rośliny) lub wykonując losowanie z zadanymi prawdopodobieństwami [3][6][7].

W przyrodzie często zdarza się, że roślina wypuszcza kwiaty po kilku słonecznych dniach lub najwcześniej dopiero po wypuszczeniu danej liczby liści. Jeśli chcemy uzyskać realistyczne efekty musimy uwzględnić te czynniki projektując nasz system.

Rozwiązaniem problemu wystąpienia jakiegoś zjawiska w przyrodzie (odpowiednia temperatura, deszcz, słoneczny dzień) jest utworzenie tabeli, która w każdej kolumnie zawiera produkcje po wystąpieniu danego zjawiska. Pomiędzy kolumnami przechodzimy wtedy albo po wykonaniu danej liczby iteracji w poprzedniej kolumnie albo losując wystąpienie zjawiska po każdej iteracji.

Rozwiązaniem problemu pojawiania się pewnych schematów z czasem (np. roślina wypuszcza kwiat po wypuszczeniu 5 liści) jest zastosowanie produkcji parametrycznych. Parametry możemy wykorzystać wtedy jako pamięć o ilości wystąpienia danego zjawiska, i dla poszczególnych produkcji zdefiniować warunki logiczne[3].

10. Rysowanie liści i kwiatów

Opisane do tej pory systemy pozwalają na tworzenie realistycznych szkieletów drzew i roślin. Jeśli chcemy generować realistyczne modele 3D to musimy zmierzyć się z problemem generowania obiektów takich jak kwiaty i liście. W tym celu należy wprowadzić do naszej interpretacji graficznej symbole umożliwiające generowanie wielokątów oraz zmiany tekstur [5]. Aby wygenerować wielokąt, nasz zbiór komend rozszerzymy o komendy wyznaczające sekcję do wypełnienia wielokątami [{}] oraz komendę zaznaczająca wierzchołek polygonu [.]. Po narysowaniu obiektu oddzielonego w odpowiedniej sekcji połączymy ze sobą wierzchołki wielokątów i utworzymy obiekt. Dodatkową przydatną komendą będzie zmiana tekstury/koloru rysowania [~].

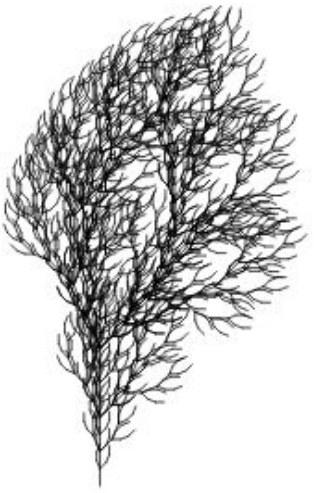
11. Nasz pomysł na implementację

Do realizacji naszego projektu zdecydowaliśmy się wykorzystać technologię Unity3D. Skrypty będą pisane w języku C#. W ostatecznej wersji aplikacji możliwe będzie zdefiniowanie elementów systemu takich jak: ciąg startowy, reguły zastępowania poszczególnych symboli, znaczenie poszczególnych znaków w interpretacji graficznej. Aplikacja będzie posiadała możliwość skorzystania z wszystkich opisanych w tej pracy opcji produkcji co zapewnia możliwość modelowania dowolnych roślin.

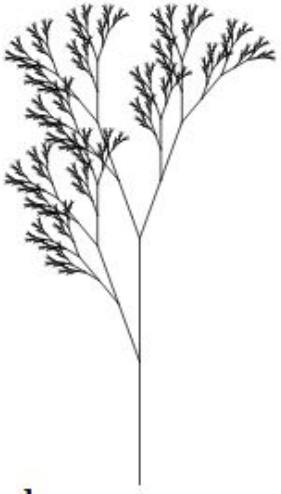
12. Przykładowe L-systemy

a
n=5, δ=25.7°
F
F → F[+F]F[-F]F

b
n=5, δ=20°
F
F → F[+F]F[-F][F]

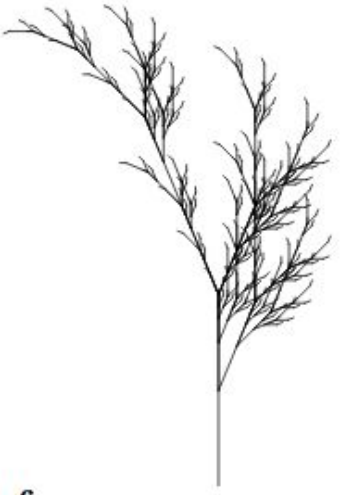


c
n=4, δ=22.5°
F
F → FF-[-F+F+F]+
[+F-F-F]



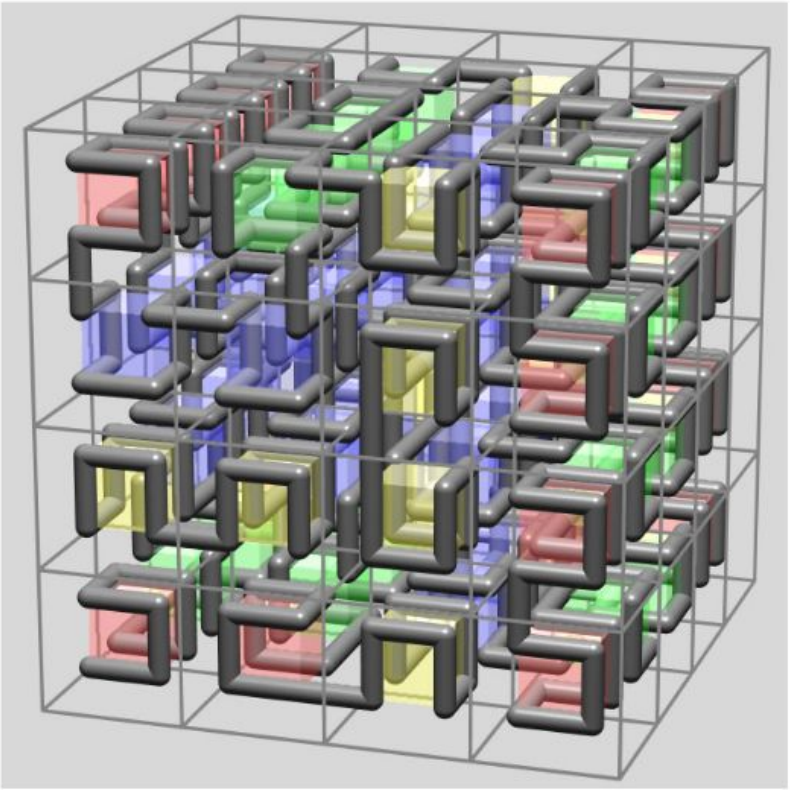
d
n=7, δ=20°
X
X → F[+X]F[-X]+X
F → FF

e
n=7, δ=25.7°
X
X → F[+X][-X]FX
F → FF



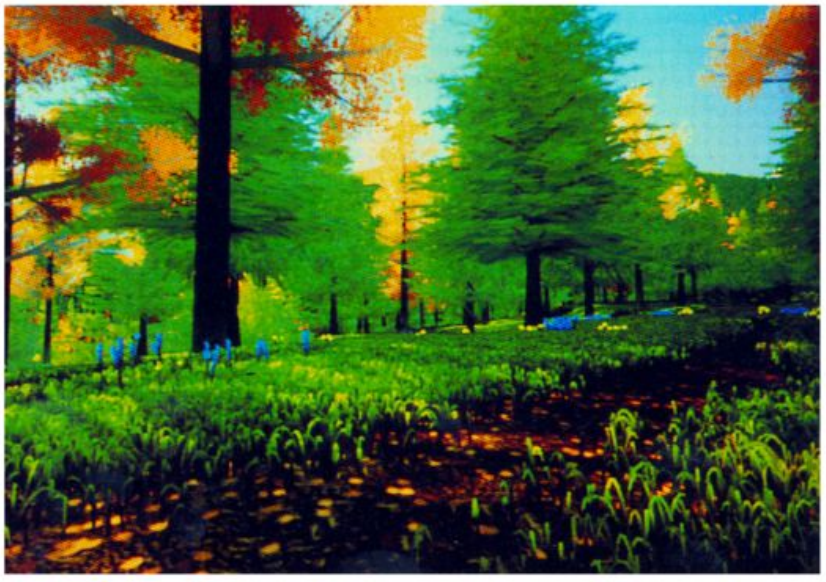
f
n=5, δ=22.5°
X
X → F-[[X]+X]+F[+FX]-X
F → FF

[7]



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//

[3]



Drzewa wygenerowane przez studio Pixar przy użyciu L-systemów

13. Bibliografia

[1] “L-System.” Wikipedia, Wikimedia Foundation, 7 Sept. 2017, pl.wikipedia.org/wiki/L-system.

[2] “L-System.” Wikipedia, Wikimedia Foundation, 28 Nov. 2017, en.wikipedia.org/wiki/L-system.

[3] Prusinkiewicz Przemyslaw, and Aristrid Lindenmayer - “The Algorithmic Beauty of Plants”, Springer-Verlag, 1996.

[4] Grzegorz Rozenberg , and Arto Salomaa - “L Systems” - 1974

[5] Mark S. Hammel, Przemyslaw Prusinkiewicz, and Brian Wyvill - “Modelling Compound Leaves Using Implicit Contours” - 1992

[6] Lisa Streit, Pavol Federl, and Mario Costa Sousa - “Modelling Plant Variation Through Growth”, EUROGRAPHICS Volume 24 (2005)

[7] Przemyslaw Prusinkiewicz, and James Hanan - “Lecture Notes in Biomathematics, Lindenmayer Systems”, 1980

2. Proponowany model zjawiska

2.1 Cele modelu:

Celem naszego modelu jest odwzorowanie generacji organizmów roślinnych jak drzewa czy krzewy, tak by wyglądały jak najbardziej naturalnie.

2.2 Założenia algorytmów:

Zastosowane przez nas algorytmy L-systemów w swoim założeniu mają za zadanie tworzenie od podstaw struktur jak najbardziej roślinopodobnych. W tym celu program interpretuje produkcję (zbiór znaków) według wcześniej zdefiniowanego słownika. Domyślnie wykorzystywany przez nas słownik to:

‘F’ - “Forward”

‘!’ - “Change width”

‘+’ - “Rotate U”

‘-’ - “Rotate U2”

‘&’ - “Rotate L”

‘^’ - “Rotate L2”

‘\’ - “Rotate H”

‘/’ - “Rotate H2”

‘$’ - “Dollar rotation”

‘[’ - “Push position”

‘]’ - “Pull position”

Istnieje jednak jest możliwość załadowania własnego słownika z pliku JSON. Taki plik powinien mieć wtedy strukturę:

```
{
	"Dictionary":[
		{
			"Letter":"F",
			"Command":"Forward",
			"Arguments":["t0"]
		},
		{
			"Letter":"!",
			"Command":"Change width",
			"Arguments":["t0"]
		}
		...
	]
}
```

Należy zadbać aby komendy na które będą tłumaczone symbole były obsługiwane przez nasz system. Obsługiwane przez system komendy to:
“Forward” - rysuj linię
“Change width” - zmień grubość linii
“Rotate U” - skręć w lewo,
“Rotate U2” - skręć w prawo,
“Rotate L” - skręć w dół,
“Rotate L2” - skręć w górę,
“Rotate H” - “przeturlaj się” w lewo
“Rotate H2” - “przeturlaj się” w prawo
“Dollar rotation” - “przeturlaj” żółwia wokół własnej osi tak, aby po lewej stronie miał podłoże
“Push position” - zapamiętaj aktualną pozycję na stosie
“Pull position” - pobierz pozycję ze stosu

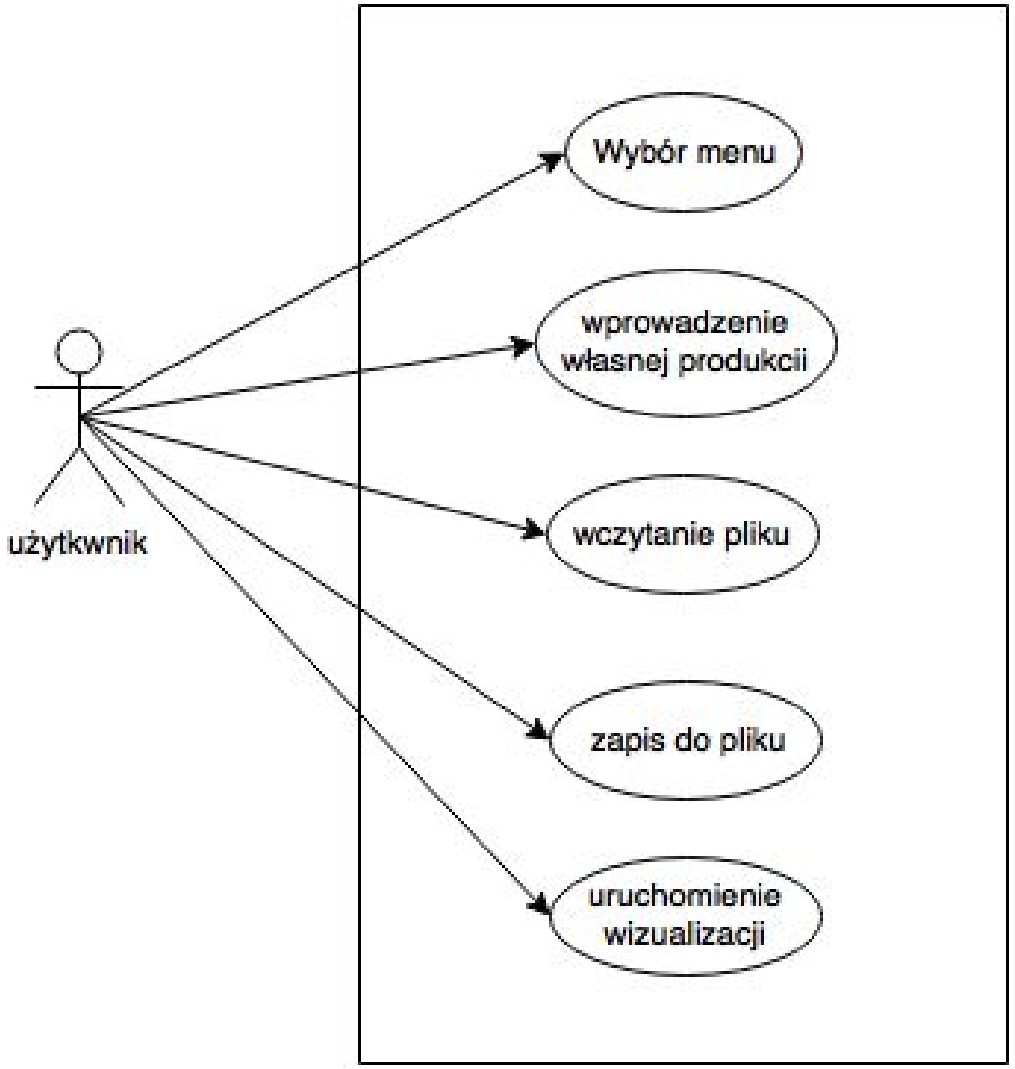
Jeżeli nasz słownik ma obsługiwać parametryczny L-system, wtedy musimy to uwzględnić w polu “Arguments”. Parametry numerujemy t0...t9.

Na podstawie przetłumaczonych przez słownik komend zostaje wygenerowany obiekt 3D, z możliwością renderowania krok po kroku, bądź natychmiastowego.

2.3 Stosowane podejście w stosunku do opisu literaturowego

Przy tworzeniu programu w znacznej mierze opieraliśmy się na opisanych we wcześniejszych rozdziałach założeniach teoretycznych, oraz stosownej ich interpretacji, zależnie od aktualnych potrzeb. Między innymi, testowaliśmy generowanie przykładowych struktur obecnych w literaturze.

2.4 Diagram przypadków użycia:



3. Symulacja zjawiska - implementacja

3.1 Wybrane technologie

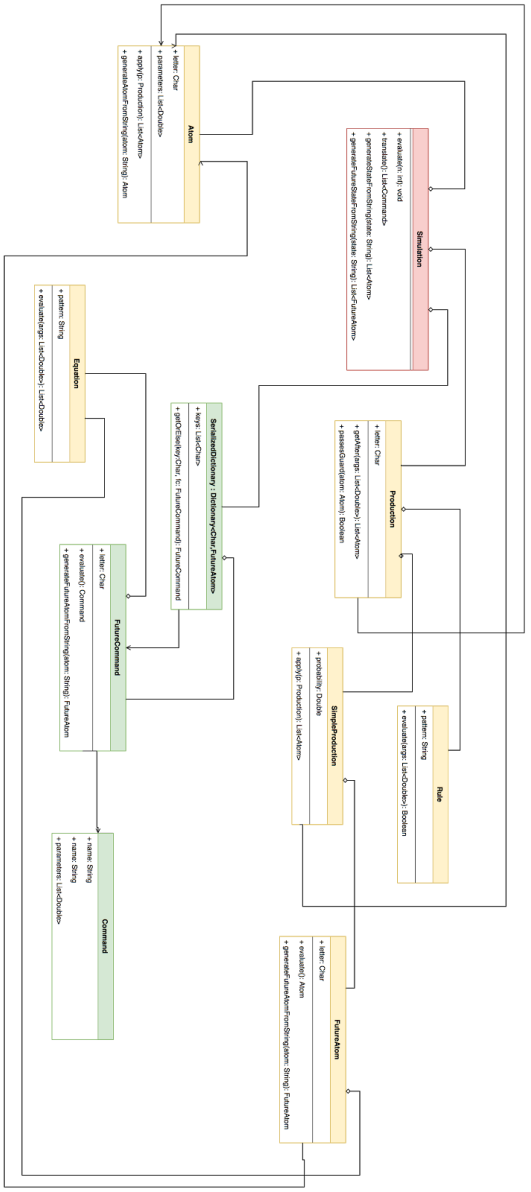
Głównym celem naszego projektu było graficzne przedstawienie L-systemów, dlatego zaszła konieczność użycia silnika graficznego. Wybraliśmy silnik Unity ze względu na to że jest jednocześnie względnie prosty w obsłudze, a efekty nim uzyskane są bardzo zadowalające. Kolejnym argumentem przemawiającym za zastosowaniem tego silnika jest duża ilość darmowych rozszerzeń. W swoim projekcie wykorzystaliśmy paczkę NatureStarterKit2 (https://www.assetstore.unity3d.com/en/#!/content/52977) aby stworzyć otoczenie dla generowanego przez nas drzewa. Dzięki temu aplikacja prezentuje się dużo korzystniej. Do ułatwienia obsługi plików wykorzystaliśmy paczkę UnityStandaloneFileBrowser (https://github.com/gkngkc/UnityStandaloneFileBrowser), gdyż Unity nie ma w sobie zaimplementowanej biblioteki umożliwiającej obsługę plików poza edytorem.

Do zaimplementowania naszego modelu zdecydowaliśmy się wykorzystać język C#, ponieważ skrypty dla silnika Unity są pisane właśnie w nim. Obie te technologie są dostępne bezpłatnie (Unity w wersji edukacyjnej).

W trakcie implementacji zaszła także potrzeba użycia biblioteki umożliwiającej ewaluowanie wyrażeń matematycznych na podstawie ich reprezentacji w zmiennej typu String. Z dostępnych bibliotek wybraliśmy NCalc (https://github.com/MichaelAguilar/NCalc), ponieważ jest ona prosta w obsłudze, a jej kod jest publicznie dostępny.

Do obsługi plików JSON wykorzystaliśmy bibliotekę JSON.Net (https://github.com/JamesNK/Newtonsoft.Json), ponieważ jest darmowa i intuicyjna w użyciu.

3.2 Schemat klas



*schemat klas w wyższej rozdzielczości został dołączony na końcu tego dokumentu

3.3 Opis implementacji

Główną klasa sterująca działaniem symulacji jest **Simulation.** Zawiera ona listę atomów tworzących jej aktualny stan, listę możliwych produkcji oraz słownik umożliwiający przetłumaczenie stanu na listę komend zrozumiałych dla części interpretującej grafikę. Klasa ta posiada metodę evaluate, która zmienia stan symulacji o n kroków, oraz translate, która tłumaczy stan na listę komend.

Klasa **Atom** odwzorowywuje pojedynczy terminal w stanie symulacji. Składa się na zmienna typu Char będąca nazwą atomu oraz listę parametrów przechowywaną przez atom.

Klasa **Production** odwzorowywuje produkcje implementujące wszystkie funkcjonalności opisane we wstępie tego dokumentu (tj. warunki i prawdopodobieństwo). Zawiera ona zmienną typu Char reprezentującą stan przed aplikowaniem produkcji, warunek aplikowania produkcji typu Rule oraz listę obiektów SimpleProduction.

Klasa **SimpleProduction** odwzorowywuje “prawą” stronę produkcji. Zawiera ona prawdopodobieństwo swojego zajścia oraz listę obiektów typu FutureAtom.

Klasa **FutureAtom** jest analogiczna do klasy Atom, z tą różnicą, że zamiast parametrów przechowuje ona algorytm otrzymania ich (lista obiektów typu Equation). Aby otrzymać Atom zgodny z tym algorytmem na obiekcie FutureAtom należy wywołać metodę evaluate(args: Double).

Klasa **Equation** reprezentuje równanie matematyczne zapisane w postaci zmiennej typu String. Używając metody evaluate(args: List<Double>) na obiekcie tej klasy można otrzymać wynik równania w postaci Double. Zmienne w równaniu muszą mieć format t0,t1,t2...t9. W trakcie ewaluacji zmienne z listy są podstawiane zgodnie z kolejnością.

Klasa **Rule** działa analogicznie do klasy Equation z tą różnicą, że metoda evaluate zwraca typ Boolean.

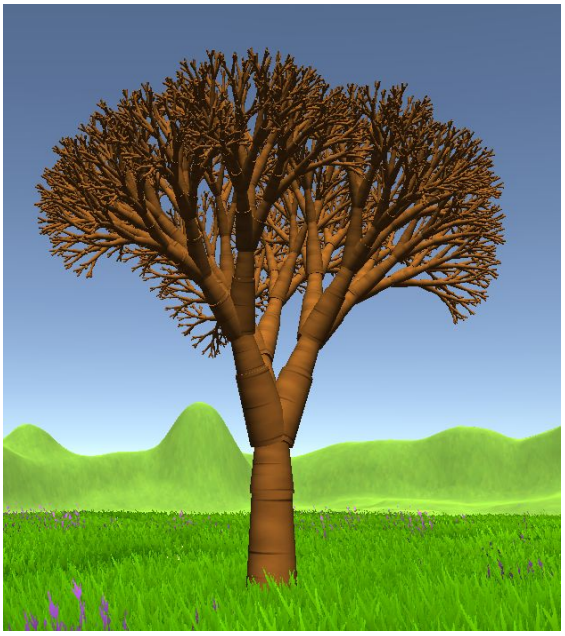
Klasa **Command** zawiera nazwę komendy typu String oraz listę parametrów typu Double. Lista obiektów tej klasy jest wykorzystywana przez obiekt sterujący rysowaniem.

Klasa **FutureCommand** implementuje analogiczną funkcjonalność jak klasa FutureAtom względem klasy Atom.

Wszystkie opisane powyżej klasy posiadają szczegółową dokumentację swojego działania. Znajduje się ona w kodzie źródłowym projektu.

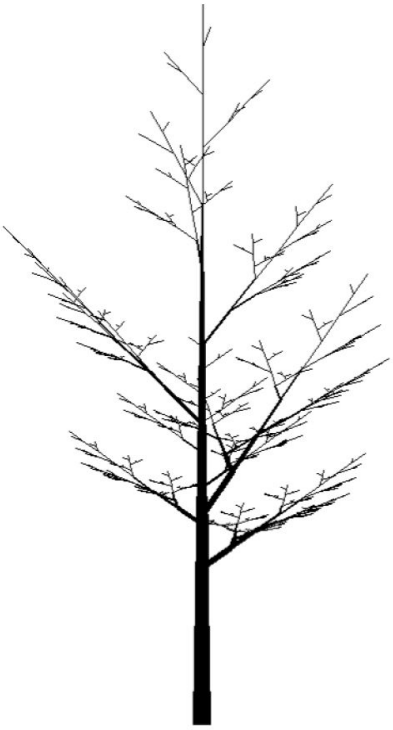
4. Wyniki Symulacji

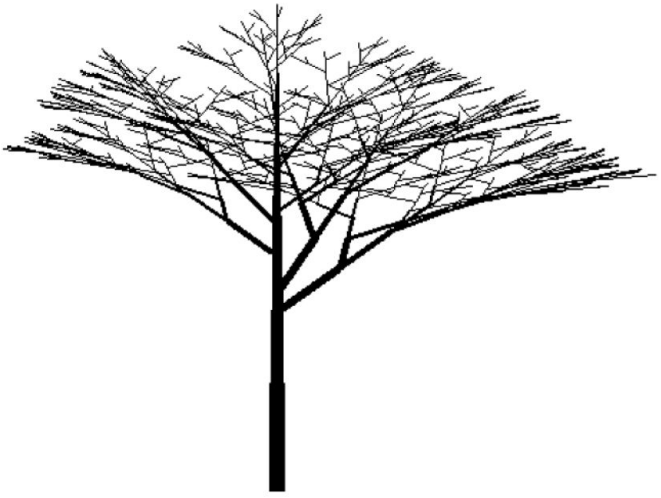
4.1 Przykładowe wyniki symulacji:

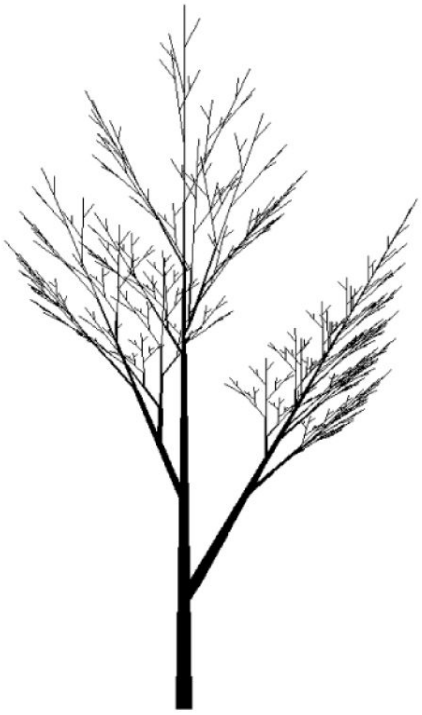


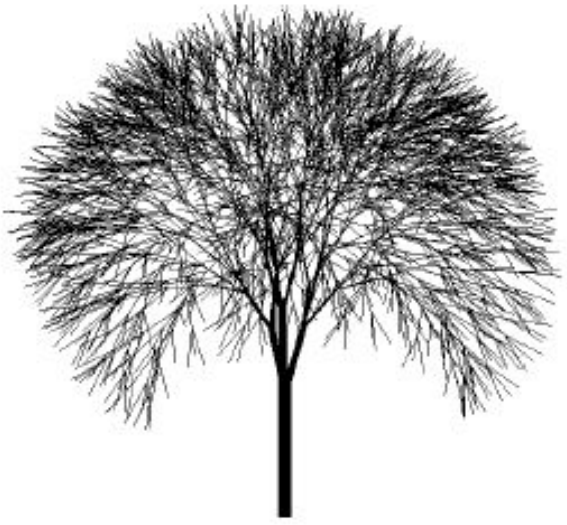
4.2 wzorce przykładowych struktur

Wyniki przykładowe były wzorowane na poniższych strukturach [3]:









4.3 Rzeczywiste drzewa bez liści

Dla porównania rzeczywiste drzewa bez liści:

Jak widać nasze przykładowe modele przypominają drzewa, ale nadal brakuje im pełnego odwzorowania rzeczywistości. Można by było popracować nad parametrami w celu osiągnięcia głębszego realizmu.

5. Wnioski

L-systemy wykazują duży potencjał w zastosowaniu we wszelkiego rodzaju programach graficznych, grach czy symulacjach. Przy dobraniu odpowiednich parametrów potrafią naprawdę wiernie odwzorować strukturę rzeczywistej rośliny, a po dobraniu odpowiednich tekstur na pierwszy rzut oka mogą być nie do odróżnienia od wzorcowego obiektu.

W naszej symulacji pierwotnie zaczęliśmy od stworzenia modeli dwuwymiarowych, by przetestować zachowanie się L-systemów w praktyce. Początkowo uzyskaliśmy zarówno proste jak i złożone struktury fraktalne. Struktury te były tworzone na zasadzie tzw. “interpretacji żółwik” o której była mowa wyżej. Zaś po dodaniu możliwości rozgałęziania się otrzymaliśmy obrazy przypominające rośliny w rzucie dwuwymiarowym.

Kolejnym ważnym krokiem była implementacja trzeciego wymiaru w procesie generacji. Umożliwiło to powstanie przestrzennych struktur drzewo podobnych. Zostały one dodatkowo udoskonalone poprzez wprowadzenie walców o różnych rozmiarach (wysokość i szerokość) które zmniejszały swój wymiar wraz z kolejnymi krokami “żółwika”(generacją kolejnych elementów struktury). Następnie walce zastąpiliśmy ściętymi stożkami, tak by finalny konstrukt wydawał się bardziej jednolity i zbliżony do rzeczywistych organizmów roślinnych.

Efekt końcowy został uwidoczniony na przykładach.

Jak dotąd nie udało nam się zaimplementować liści na generowanych drzewach. Co mogłoby stanowić możliwość dalszego rozwoju aplikacji. Również można by było pokusić się o generację dodatkowych modeli roślin np. ziół czy traw.