



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 | 4 |
| 1 System Lindenmayera | 5 |
| 1.1 Informacje wstępne | 5 |
| 1.2 Struktura L-systemu | 6 |
| 1.3 Interpretacja ciągu znaków | 9 |
| 1.3.1 Trójwymiarowe generowanie drzew | 13 |
| 2 Implementacja | 15 |
| 2.1 Wykorzystane narzędzia | 15 |
| 2.1.1 Język i zintegrowane środowisko programistyczne | 15 |
| 2.1.2 Biblioteka Proctree | 15 |
| 2.1.3 Biblioteka Nlohmann JSON | 16 |
| 2.1.4 OpenGL | 16 |
| 2.2 Funkcjonalność aplikacji | 16 |
| 2.2.1 Planowanie rozmieszczenia drzew | 17 |
| 2.2.2 Symulacja wzrostu | 19 |
| 2.2.3 Ustawianie parametrów | 21 |
| 2.2.4 Ustawianie tekstur | 26 |
| 2.2.5 Zapis do pliku | 29 |
| 2.3 Struktura programu | 30 |
| 3 Testy i wyniki | 33 |
| 3.1 Wydajność | 34 |
| 3.2 Porównanie z innymi rozwiązaniami | 35 |

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%), tworzy 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 doczekała się opisanego matematycznie modelu 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 Lindenmayera, który umożliwia opis struktur fraktalnych, w szczególności roślin na poziomie gramatyki formalnej.

Celem pracy jest zastosowanie i ewaluacja systemu Lindenmayera do generowania roślin w aplikacji do planowania przestrzeni zielonych. Specyfikacja programu została przygotowana we współpracy z pracownikami Instytutu Nauk Biologicznych Uniwersytetu Marii Curie Skłodowskiej w Lublinie. 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

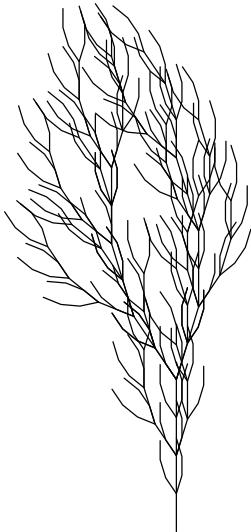
System Lindenmayera

1.1 Informacje wstępne

Systemy Lindenmayera (L-Systemy) zostały wprowadzone i rozwinięte w 1968 roku przez Aristida Lindenmayera [2], 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 [3].

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



Rysunek 1.1: Przykład stworzonej struktury za pomocą L-systemu

1.2 Struktura L-systemu

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

$$G = (V, \omega, P), \quad (1.1)$$

gdzie

- V (alfabet) – to zbiór symboli zawierający zarówno elementy, które można zastąpić (zmienne), 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 znaków (ewentualnie pustym) w tym samym alfabetie. 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.

Tabela 1.1: Przykład stworzonej struktury za pomocą L-systemu

| Alfabet | Aksjomat | Reguły |
|---|----------|---|
| $\{ 'A', 'B', 'F', 'H', 'J', '+', '-' \}$ | $'FB'$ | $'A' \rightarrow 'FBFA + FB - FA'$ $'B' \rightarrow 'FB + FA - FB - J'$ $'F' \rightarrow '$ $'H' \rightarrow '-'$ $'J' \rightarrow '+'$ |

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 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 $'X'$, dla których nie istnieje reguła jawną, obowiązuje reguła domyślna: $'X' \rightarrow 'X'$. Jeśli zostanie znaleziona pasująca reguła, znak poprzednika jest zastępowany przez łańcuch następnika z tej reguły.

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

Tabela 1.2: Przykład stworzonej struktury za pomocą L-systemu

| Aksjomat | Reguły |
|----------|----------------------------|
| $^r A_$ | $^r A_ \rightarrow ^r B_$ |
| | $^r B_ \rightarrow ^r AB_$ |

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

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

| Generacja | Stan |
|-----------|---------------------------------------|
| 0 | $^r A_$ |
| 1 | $^r B_$ |
| 2 | $^r AB_$ |
| 3 | $^r BAB_$ |
| 4 | $^r BABAB_$ |
| 5 | $^r BABABBAB_$ |
| 6 | $^r BABABBABABBAB_$ |
| 7 | $^r BABABBABABBABBABABBAB_$ |
| 8 | $^r BABABBABABBABBABABBABABBABABBAB_$ |

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. Funkcja zwraca stan łańcucha po podanej liczbie iteracji.

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

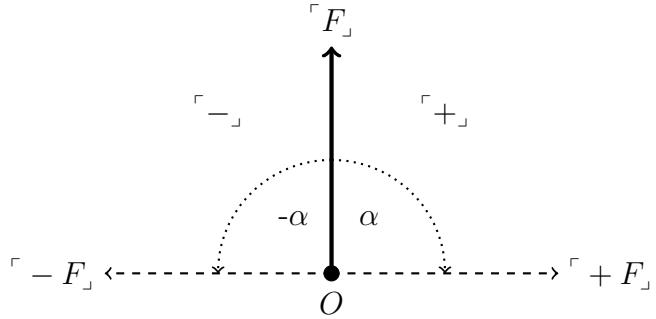
```
1 def iter(axiom: str, rules: dict, iterations: int) -> str:
2     if iterations == 0: return axiom
3     returnString = ''
4     for i in axiom:
5         if i in rules:
6             returnString += rules[i]
7         else:
8             returnString += i
9     return iter(returnString, rules,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

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. Nawiązanie do żółwia inspirowane jest powstały w latach 60. językiem Logo, służącym do nauki programowania, który wykorzystywał grafikę żółwia jako dwuwymiarowy kurSOR.

Interpretacja znaków polega na zdefiniowaniu operacji dla symboli (nie jest konieczne dla wszystkich) w alfabetie. 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:

- 「 F 」 oznacza przejście do przodu i narysuje linię,
- 「 $-$ 」 oznacza obrót w kierunku przeciwnym do ruchu wskazówek zegara na kąt o mierze α ,
- 「 $+$ 」 oznacza obrót zgodnie z ruchem wskazówek zegara o α .



Rysunek 1.2: Przykładowa interpretacja symboli

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

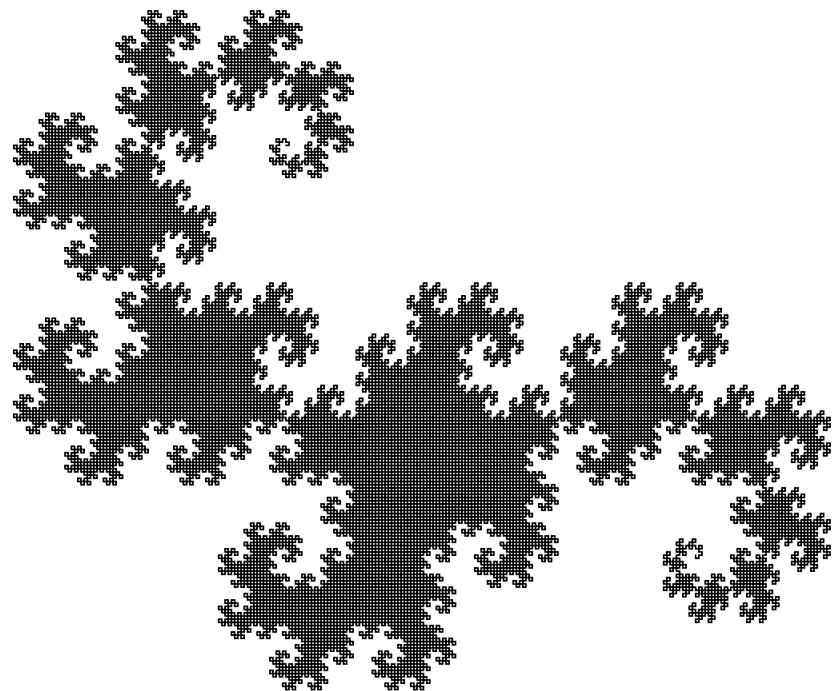
Widać, że po każdej iteracji struktura staje się coraz bardziej złożona. Wynik piętnastu iteracji pokazano na rysunku 1.3.

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

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

Tabela 1.5: Krzywa smoka w iteracji 1-7

| Iteracji | Stan |
|----------|---|
| 1 | „L” |
| 2 | „L”-wzór |
| 3 | „L”-wzór z dodatkowymi elementami |
| 4 | „L”-wzór z dodatkowymi elementami i zmianą kształtu |
| 5 | „L”-wzór z dodatkowymi elementami i zmianą kształtu, z dodatkowymi elementami |
| 6 | „L”-wzór z dodatkowymi elementami i zmianą kształtu, z dodatkowymi elementami i zmianą kształtu |
| 7 | „L”-wzór z dodatkowymi elementami i zmianą kształtu, z dodatkowymi elementami i zmianą kształtu, z dodatkowymi elementami i zmianą kształtu |



Rysunek 1.3: Wynik piętnastu iteracji krzywej smoka

W przykładzie krzywej smoka (rysunek 1.3) żółw zawsze porusza się o tę samą odległość, a jego linie mają tę samą grubość. Jednak w świecie rzeczywistym rośliny i drzewa mają zasadniczo strukturę rozgałęzioną. W strukturze fraktalnej rośliny 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) demonstruje przykład tworzenia struktury przypominającej drzewo [4]. W przykładzie tego L-systemu do alfabetu wprowadzane są nowe znaki. Ich interpretacja przez żółwia jest następująca:

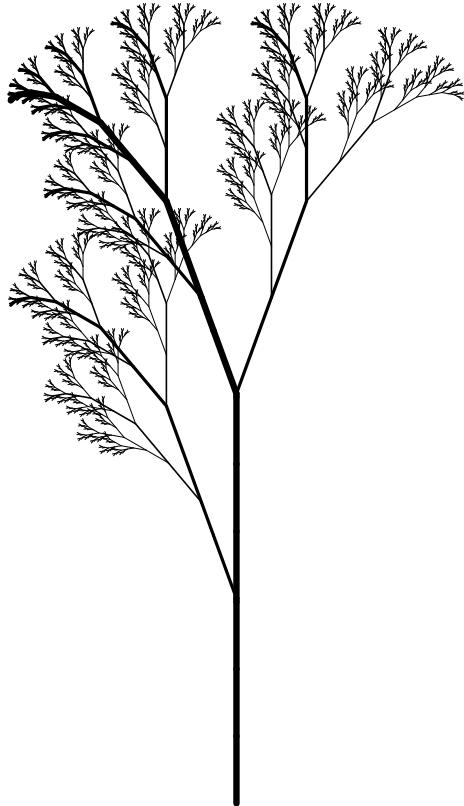
- $['[']$ oznacza “zapisać bieżący stan żółwia do stosu”,
- $['']'$ oznacza “odczytać bieżący stan żółwia do stosu”,
- $['X']'$ jest ignorowany przez żółwia.

W przykładzie przedstawionym w książce [4] 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 |
|----------------------------|----------|---|---|
| $\{ 'F', 'X', '[', ']' \}$ | $'X'$ | $'X' \rightarrow 'F[+X]F[-X] + X'$ $'F' \rightarrow 'F'$ | Kąt powrotu = 20° Grubość linii = 6px Czynnik grubości = 0.5 |



Rysunek 1.4: Utworzona struktura na podstawie reguł z tabeli 1.6

1.3.1 Trójwymiarowe generowanie drzew

Aby utworzyć trójwymiarowe drzewa, należy zdefiniować symbole dla żółwia, który będzie odpowiedzialny za obroty w przestrzeni trójwymiarowej. Jednym ze sposobów byłoby zdefiniowanie obrotów żółwia na każdej osi (X, Y, Z), a także możliwość ustalenia kąta obrotu. Dzięki temu żółw może poruszać się we wszystkich możliwych kierunkach. Podczas budowania drzewa, w momencie gdy gałąź się rozgałęzia, należy, podobnie jak w przypadku 2D, ustawić kąty dla każdej z nowych gałęzi. Lindenmayер sugeruje użycie trzech wektorów do reprezentowania kierunku żółwia (przód, lewo i góra) [4].

W tym przypadku obroty żółwia będą reprezentowane przez następujące równanie:

$$[\vec{H}' \vec{L}' \vec{U}'] = [\vec{H} \vec{L} \vec{U}] R, \quad (1.3)$$

gdzie

- H jest wektorem wskazującym góre żółwia,
- L jest wektorem wskazującym lewą stronę żółwia,
- F jest wektorem wskazującym przód żółwia,
- R jest macierzą trójwymiarową.

Macierz R, w zależności od osi obrotu, może wyglądać w następujący sposób:

$$R_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, \quad (1.4)$$

$$R_L(\alpha) = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{bmatrix}, \quad (1.5)$$

$$R_U(\alpha) = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.6)$$

Rozdział 2

Implementacja

W zakresie implementacji stworzyłem oprogramowanie, które pozwala na wizualizację przyległego terenu do nieruchomości w trójwymiarowej przestrzeni, na której można umieścić drzewa. Drzewa można zmieniać za pomocą różnych parametrów i symulować ich wzrost.

2.1 Wykorzystane narzędzia

2.1.1 Język i zintegrowane środowisko programistyczne

Do stworzenia programu wykorzystałem język programowania C++ w standardzie 17. Użyłem Qt Creator w wersji 6.0.1 jako zintegrowanego środowiska programistycznego.

Stworzone oprogramowanie wykorzystuje język C++ dla całej logiki aplikacji.

2.1.2 Biblioteka Proctree

Proctree.js to biblioteka dla języka programowania JavaScript, która pozwala na tworzenie drzewa jako siatki wierzchołków [5]. Biblioteka posiada dużą liczbę parametrów, które wpływają na ostateczny wynik drzewa. Autorem biblioteki jest brytyjski programista Paul Brunt. Później fiński programista Jari Komppa zrobił port tej biblioteki dla C++. Biblioteka w wersji C++ jest szybsza [6].

Biblioteka Proctree jest głównym narzędziem do tworzenia siatki wierzchołków, normalnych i mapowania UV dla drzewa w programie.

2.1.3 Biblioteka Nlohmann JSON

Nlohmann JSON to biblioteka do obsługi danych formatu JSON [7]. Biblioteka ma wiele zalet: intuicyjna syntaktyka, łatwa integracja, wydajność pamięci, szybkość. Autorem biblioteki jest niemiecki programista Niels Lohmann.

Biblioteka Nlohmann JSON jest używana w programie do zapisywania i ładowania parametrów drzewa w postaci pliku JSON.

2.1.4 OpenGL

OpenGL jest specyfikacją definiującą niezależny od platformy interfejs programowania aplikacji do pisania aplikacji wykorzystujących grafikę komputerową 2D i 3D. Twórcą jest amerykańska firma Silicon Graphics. OpenGL pozwala na tworzenie złożonych scen 3D za pomocą prymitywów. Wykorzystuje do tego zasoby karty graficznej. Wykorzystywany jest do tworzenia gier komputerowych, wirtualnej rzeczywistości.

Powstałe oprogramowanie wykorzystuje OpenGL do renderowania trójwymiarowej przestrzeni i drzew. Zostało przetestowane w wersji OpenGL 3.3.

2.2 Funkcjonalność aplikacji

Stworzone oprogramowanie posiada szereg funkcji, które pozwalają na zaprojektowanie rozmieszczenia drzew w otoczeniu przy budynku.

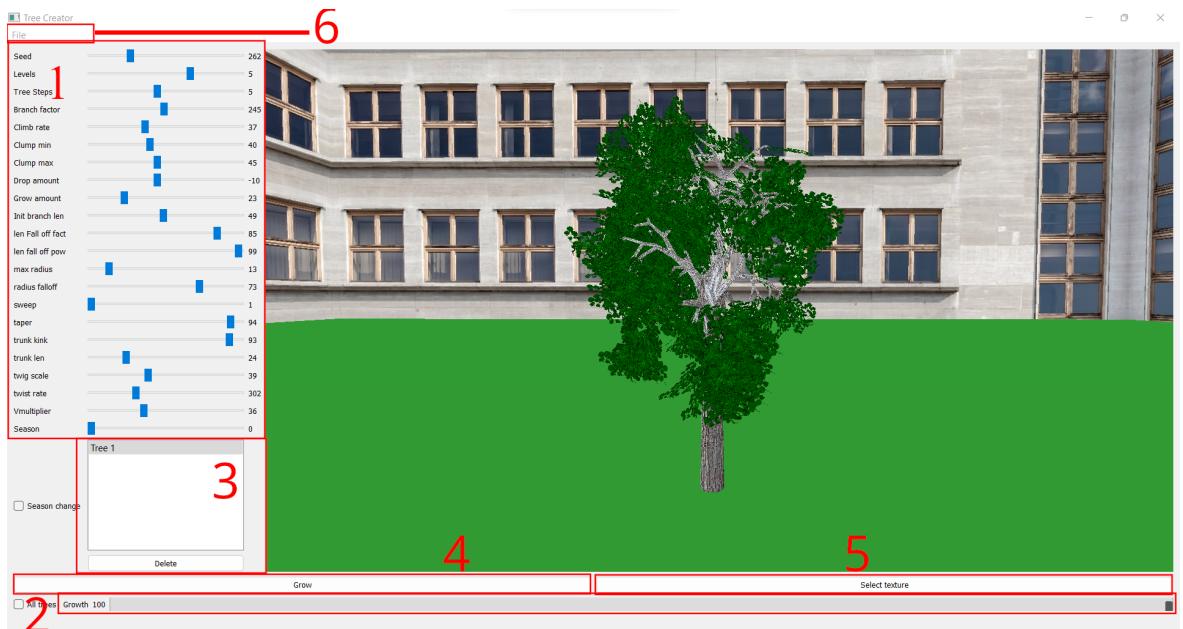
Na rysunku 2.1 można zaobserwować główne okno programu. Konwencjonalnie okno główne można podzielić na dwa obszary – obszar ustawień i obszar roboczy.

Część ustawień znajduje się po lewej i dolnej stronie okna. Obszar roboczy zajmuje większość okna i znajduje się po prawej stronie okna.

Obszar ustawień obejmuje:

- 1) suwak do regulacji drzew,
- 2) suwak do zmiany wieku drzewa,

- 3) lista utworzonych drzew,
- 4) przycisk symulacji wzrostu drzew,
- 5) przycisk do zmiany tekstur,
- 6) pasek menu.



Rysunek 2.1: Główne okno programu

Obszar roboczy to trójwymiarowa przestrzeń, która symuluje obszar wokół budynku drzewa. Kamerę można przesuwać za pomocą klawiatury. Klawisze W, A, S, D służą do poruszania się odpowiednio do przodu, w lewo, do tyłu i w prawo. Klawisz Q służy do poruszania się wyżej. Aby obniżyć kamerę niżej, użyj klawisza Z. Kamerę można obracać za pomocą myszy z wcisniętym lewym przyciskiem.

2.2.1 Planowanie rozmieszczenia drzew

Aby posadzić drzewa, najpierw za pomocą myszy i klawiatury ustawiamy kamerę tak, aby widoczny był obszar, w którym chcemy posadzić drzewko (rysunek 2.2). Następnie umieszczamy kurSOR w miejscu, w którym chcemy posadzić drzewo i klikamy prawym przyciskiem myszy (rysunek 2.3).



Rysunek 2.2: Okno programu przed posadzeniem drzewa



Rysunek 2.3: Okno programu po posadzeniu drzewa

Nie ma ograniczeń co do liczby drzew w programie, ale należy pamiętać, że duża liczba drzew może wpływać na wydajność, a w szczególności na symulację wzrostu. Rysunek 2.4 pokazuje 50 posadzonych drzew.



Rysunek 2.4: 50 posadzonych drzew

2.2.2 Symulacja wzrostu

Na potrzeby demonstracji ustawiłem trzy drzewa i zmieniłem ich parametry, aby były różne (rysunek 2.5).



Rysunek 2.5: Trzy różnych drzewa

Symulacja wzrostu może odbywać się na dwa sposoby: ręcznie, poprzez ustawie-

nie "wieku" drzewa (rysunek 2.6, 2.7), oraz automatycznie, poprzez naciśnięcie przycisku "Rosnąć" (ang. *Grow*) (rysunek 2.8). Każda z tych metod ma dwa tryby: zmiana wzrostu pojedynczego wybranego drzewa (rysunek 2.6) lub zmiana wzrostu wszystkich drzew (rysunek 2.7, 2.8). Tryb jest przełączany za pomocą przycisku wyboru "Wszystkie drzewa" (ang. *All trees*).



Rysunek 2.6: Ręczna zmiana wzrostu jednego drzewa



Rysunek 2.7: Ręczna zmiana wzrostu wszystkich drzew



Rysunek 2.8: Automatyczny wzrost drzew

Liczba drzew wpływa na to, jak szybko będą generowane. Dzieje się tak dlatego, że każde drzewo jest generowane inaczej, biorąc pod uwagę to, jak będzie wyglądało pod koniec wzrostu. Nie udało mi się tego zoptymalizować.

Podczas zmiany wzrostu można również poruszać kamerą, ale ponieważ proces wzrostu wymaga dość dużej wydajności, kamera może poruszać się wolniej.

2.2.3 Ustawianie parametrów

Parametry w moim programie pozwalają na manipulowanie wyglądem. W efekcie mogą powstawać różne rodzaje drzew. Każdy z parametrów wpływa na określone cechy drzew. W moim programie występują następujące rodzaje parametrów:

- ziarno (*ang. seed*) to parametr odpowiedzialny za generator drzewa pseudolosowego,
- poziomy (*ang. levels*) to parametr odpowiadający za liczbę rozgałęzień głównych gałęzi (które wyrastają z pnia) drzewa (rysunek 2.9),
- stopnie drzew (*ang. tree steps*) to parametr odpowiadający za liczbę gałęzi drzewa (pnia) (rysunek 2.10),

- współczynnik gałęzi (*ang. branch factor*) to parametr wpływający na kąt rozgałęzienia wszystkich gałęzi z wyjątkiem gałęzi głównych,
- tempo wznoszenia (*ang. climb rate*) to parametr odpowiedzialny za długość pnia po miejscach, w których następuje rozgałęzienie (rysunek 2.11),
- kępa minimalna (*ang. clump min*) i kępa maksymalna (*ang. clump max*) to parametry związane wpływające na rozprzestrzenianie się wzrostu gałęzi drzew,
- wartość opadania (*ang. drop amount*) to parametr wpływający na siłę ciężkości na gałęziach drzewa (rysunek 2.12),
- tempo wzrostu (*ang. grow amount*) to parametr wpływający na siłę grawitacji dla głównych gałęzi drzewa,
- początkowa długość gałęzi (*ang. initial branch length*) to parametr odpowiedzialny za początkową długość głównych gałęzi (ponieważ kolejne gałęzie pobierają procent długości z gałęzi nadzędnej, ma to wpływ na wszystkie gałęzie) (rysunek 2.13),
- współczynnik zmniejszania długości (*ang. length fall off factor*) i moc zmniejszania długości (*ang. length fall off power*) to parametry związane wpływające na długość wszystkich gałęzi,
- maksymalny promień (*ang. max radius*) parametr określający promień drzewa (dotyczy zarówno gałęzi jak i pnia),
- spadek promienia (*ang. radius falloff*) to parametr określający grubość gałęzi po rozgałęzieniu (gałęzie pobierają pewien procent grubości gałęzi macierzystej, parametr ten ustala ten procent),
- zamiatanie (*ang. sweep*) to parametr, który wpływa na ogólny kierunek wszystkich gałęzi. Tworzy efekt podobny do tego, jak drzewo jest pod wpływem wiatru,

- stożek (*ang. taper*) to parametr wpływający na stosunek grubości gałęzi na początku do końca (przy wyższych parametrach gałąź zaczyna przypominać stożek),
- wygięcie pnia (*ang. trunk kink*) to parametr wpływający na krzywiznę pnia w miejscach, gdzie on się rozgałęzia,
- długość pnia (*ang. trunk length*) to parametr wpływający na długość pnia (dotyczy tylko części pnia przed pierwszą gałęzią) (rysunek 2.14),
- skala gałęzi (*ang. twig scale*) to parametr, który wpływa na wielkość gałązek,
- skrętność (*ang. twist rate*) to parametr odpowiadający za skręt pnia,
- mnożnik (*ang. Vmultiplier*) to parametr wpływający na rozciąganie tekstury,
- sezon (*ang. season*) to parametr, który wpływa na kolor liści drzewa (rysunek 2.15).



Rysunek 2.9: Zmiana parametru levels



Rysunek 2.10: Zmiana parametru tree steps



Rysunek 2.11: Zmiana parametru climb rate



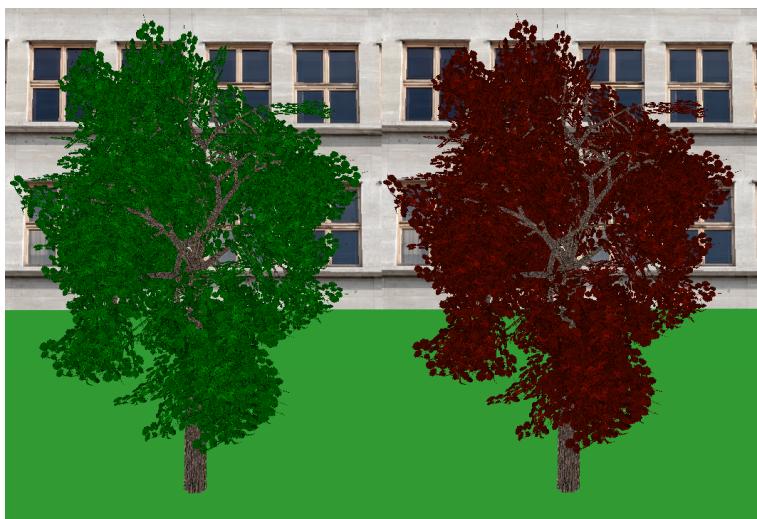
Rysunek 2.12: Zmiana parametru drop amount



Rysunek 2.13: Zmiana parametru initial branch length



Rysunek 2.14: Zmiana parametru trunk length



Rysunek 2.15: Zmiana parametru season

2.2.4 Ustawianie tekstur

Moje oprogramowanie pozwala również na zmianę tekstur. Można zmienić teksturę pnia drzewa, gałęzi drzewa i fasady budynku. Użytkownik może wybrać tekstury, które chce zastosować.

Aby zmienić teksturę pnia i gałęzi drzewa, program odczytuje pliki znajdujące się

w folderze *textures/* w korzeniu programu. Folder *textures/twigs/* zawiera tekstury gałęzi, a folder *textures/trees/* zawiera tekstury pnia. Po dodaniu tekstur do tego folderu, po uruchomieniu programu i naciśnięciu przycisku wybierz teksturę (*ang. select texture*) otwiera okno wyboru tekstury (rysunek 2.16), w którym można wybrać teksturę gałęzi i pnia drzewa z listy wszystkich tekstur znajdujących się w wyżej wymienionych folderach.



Rysunek 2.16: Okno wyboru tekstury

Po wybraniu tekstur i potwierdzeniu przyciskiem OK, tekstury drzew zostają zmienione na wybrane tekstury (rysunek 2.17).



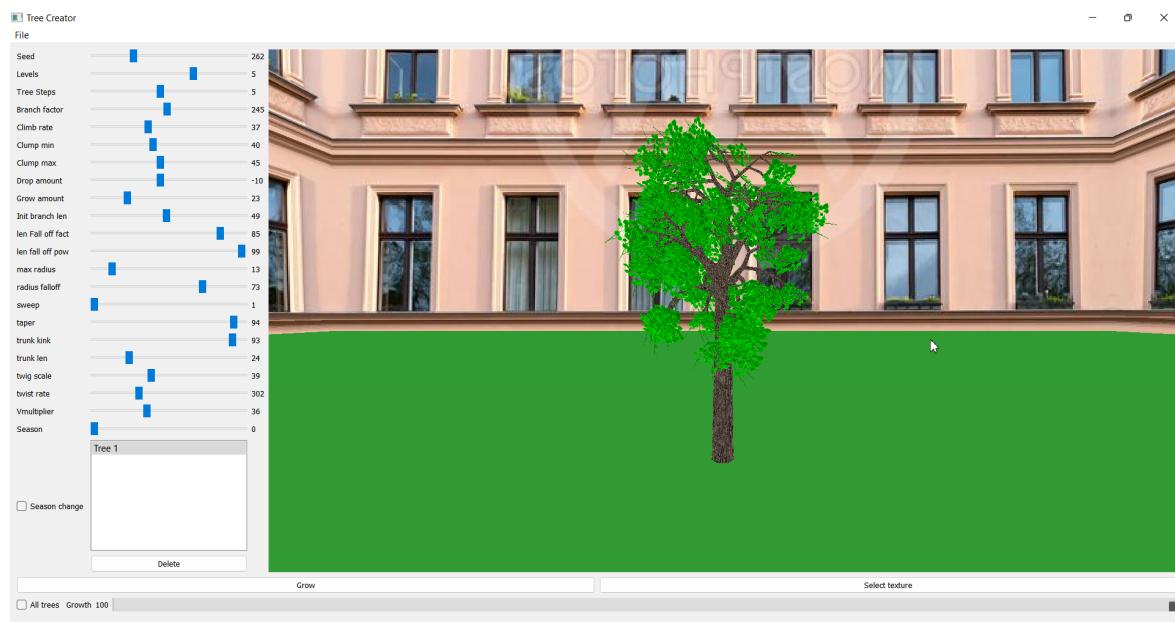
Rysunek 2.17: Zmienione tekstury drzewa

Aby zmienić teksturę budynku należy wybrać w pasku menu plik (*ang. file*), a na-

stępnie kliknąć na otworzyć teksturę budynku (*ang. open building texture*) (rysunek 2.18). Następnie zostanie otwarte okno wyboru pliku systemu operacyjnego. Za pomocą tego okna wybieramy plik, który ma być użyty jako tekstura i naciśkamy przycisk OK. Spowoduje to zmianę tekstury budynku (rysunek 2.19).



Rysunek 2.18: Zmiana tekstury budynku



Rysunek 2.19: Zmieniona tekstura budynku

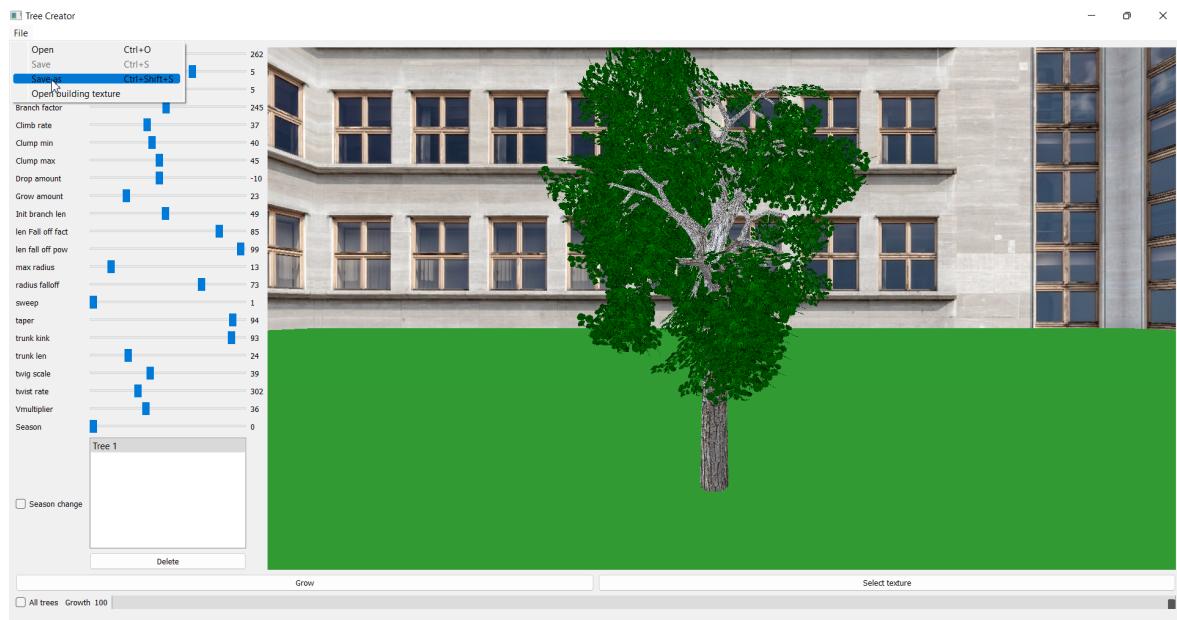
2.2.5 Zapis do pliku

W moim programie istnieje możliwość zapisania konfiguracji drzewa do pliku. Funkcja ta pozwala na zapisanie utworzonych drzew, aby można je było wykorzystać później. Aby to zrobić, trzeba będzie załadować wcześniej zapisaną konfigurację z pliku. Sam plik jest plikiem JSON (listing 2.1), w którym zapisane są wszystkie parametry.

Listing 2.1: Zawartość pliku drzewa

```
1 {
2     "mBranchFactor": 2.45,
3     "mClimbRate": 0.37,
4     "mClumpMax": 0.45,
5     "mClumpMin": 0.40,
6     ...
7     "mTwigScale": 0.38,
8     "mTwistRate": 3.019,
9     "mVMultiplier": 0.36
10 }
```

Aby zapisać wybrane drzewo, w menu paska wybierz opcję Zapisz jako (*ang. Save as*) (rysunek 2.20) lub naciśnąć kombinację klawiszy Ctrl+Shift+S. Następnie należy wybrać katalog, w którym ma być zapisany plik, wpisać jego nazwę i kliknąć przycisk Zapisz.

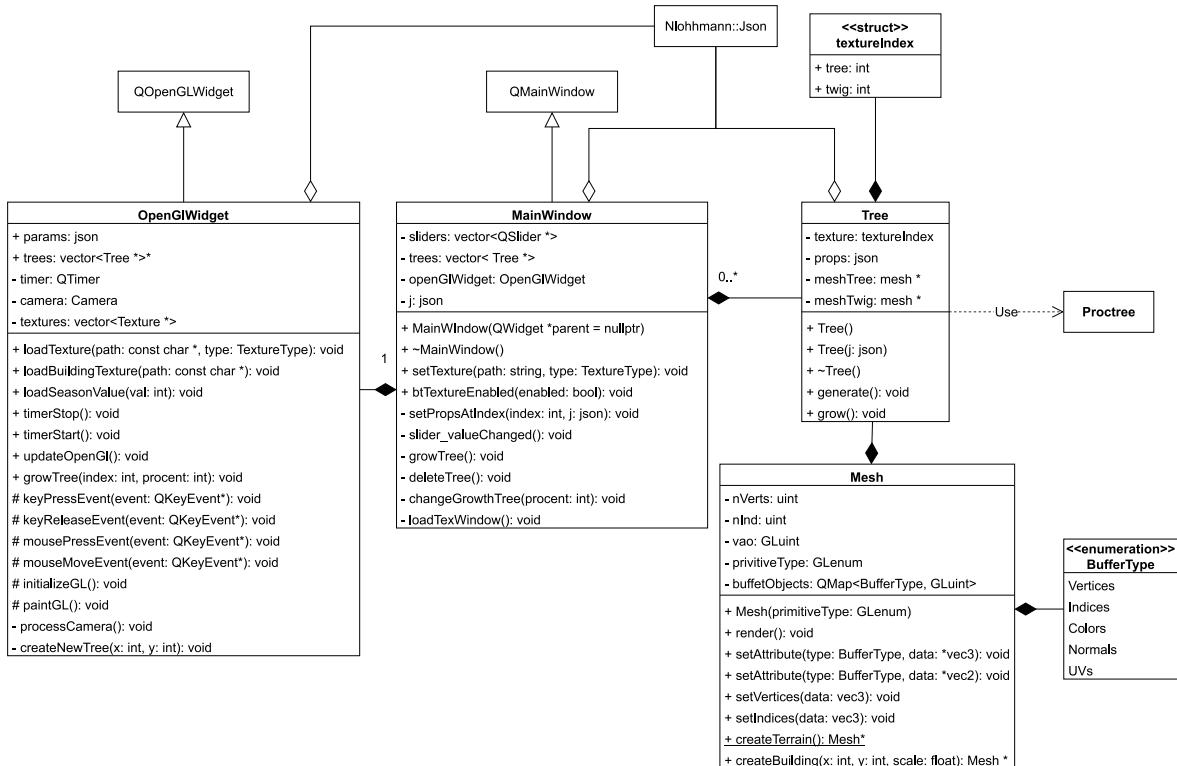


Rysunek 2.20: Opcja zapisu drzewa do pliku

2.3 Struktura programu

Struktura pozwala zademonstrować główne części programu, które wchodzą ze sobą w interakcje. Na rysunku 2.21 przedstawiono diagram klas utworzony w języku opisu graficznego UML. Większość pól klasy prywatnej posiada settery i gettery, ale dla uproszczenia nie są one pokazane na diagramie. Główne klasy w programie to:

- MainWindow ta klasa jest główną klasą programu, opisuje logikę dla wszystkich komponentów okna programu, jak również wszystkie interakcje programu,
- OpenGLWidget to klasa opisująca logikę renderowania przestrzeni trójwymiarowej,
- Tree to klasa, która reprezentuje drzewa w programie. Klasa ta przechowuje logikę zmiany wzrostu drzewa, parametry, a także dwa obiekty klasy mesh - dla pnia i gałęzi drzewa oraz liści,
- Mesh to klasa przechowująca wszystkie dane do budowy obiektów trójwymiarowych w środowisku opengl (wierzchołki, krawędzie, normale i mapy UV).

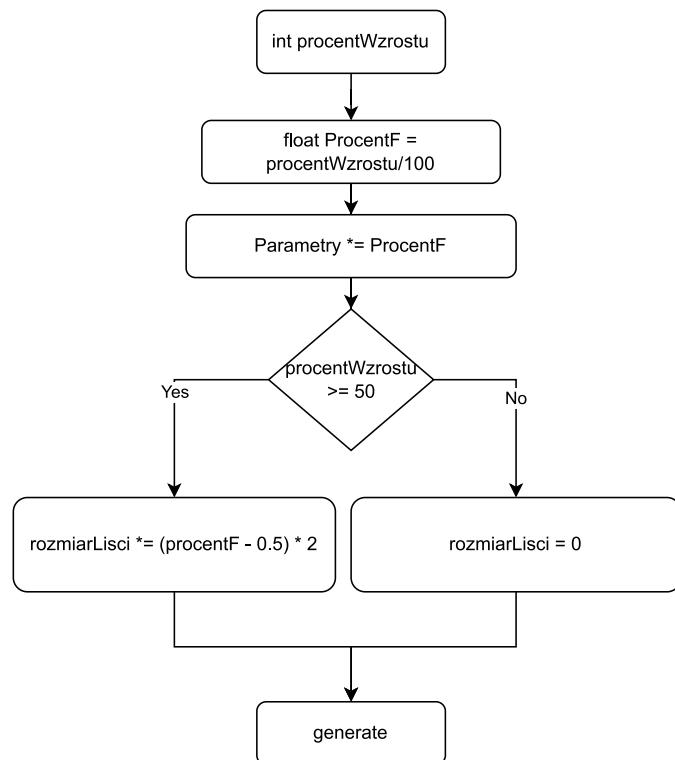


Rysunek 2.21: Diagram klasy

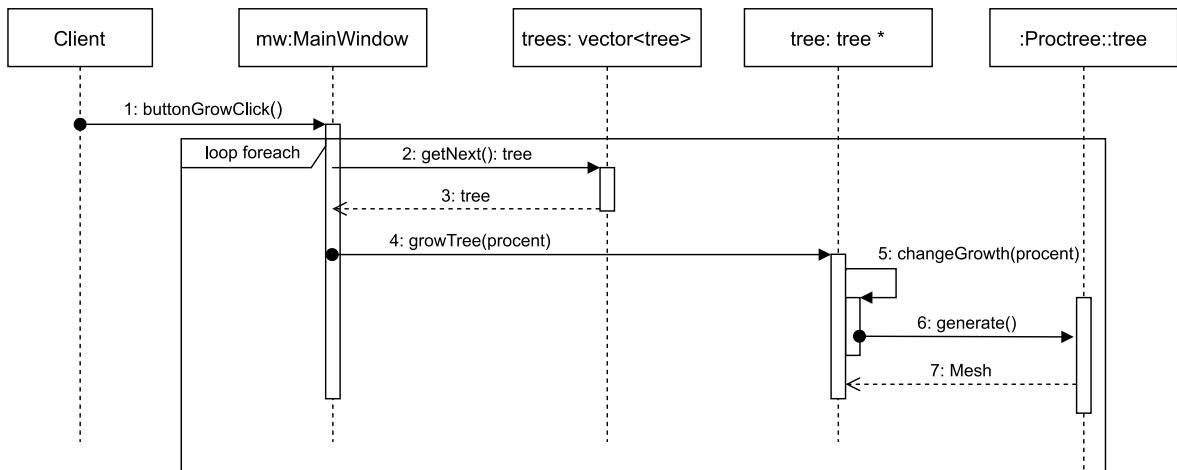
W celu zademonstrowania głównej funkcji programu (symulacja wzrostu drzew), wykonałem dwa diagramy:

- diagram procesu w celu przedstawienia logiki wzrostu drzew (rysunek 2.22),
- diagram sekwencyjny mający na celu zademonstrowanie współdziałania obiektów klas programu (rysunek 2.23).

Na diagramie sekwencji widać, jak obiekt klasy MainWindow otrzymuje informację o tym, że użytkownik nacisnął przycisk Grow. Po tym zdarzeniu obiekt mainWindow przechodzi przez każdy obiekt w drzewie i na każdym z nich wywołuje metodę growTree. Metoda growTree jest opisana na schemacie procesu. Na końcu tej metody wywoływana jest metoda generate zewnętrznej biblioteki Proctree, której klasa obsługuje generowanie i zwraca obiekt Mesh. Obiekt Mesh jest później używany do renderowania drzewa w oknie OpenGl.



Rysunek 2.22: Diagram procesu



Rysunek 2.23: Diagram sekwencji

Rozdział 3

Testy i rezultaty

Po stworzeniu, ręcznie przetestowałem stworzony program, aby sprawdzić operatywność i wydajność programu. Testy zostały przeprowadzone na laptopie o następujących konfiguracjach:

- system operacyjny Windows 10,
- procesor centralny AMD Ryzen 5 5600H 3.30 Ghz,
- karta graficzna NVIDIA GeForce GTX 1650,
- pamięć główna 16 GB.

Dzięki opracowaniu programu w środowisku Qt, program jest cross-platformowy i nie jest przywiązyany do konkretnego systemu operacyjnego, sprzętu. Ponadto, aby zainstalować oprogramowanie wystarczy rozpakować archiwum z oprogramowaniem w dowolnym miejscu na dysku twardym. Program wykorzystuje ścieżki względne, dzięki czemu może pracować z dowolnego folderu na dysku twardym. Cechy te są istotne ze względu na plan udostępnienia aplikacji szerszemu gronu odbiorców.

Testowanie programu odbywało się w trakcie tworzenia programu, jak również testowania ostatecznej wersji programu.

Testy wszystkich funkcjonalności dały pozytywny wynik, a wszystkie funkcje programu działają zgodnie z pierwotnymi warunkami.

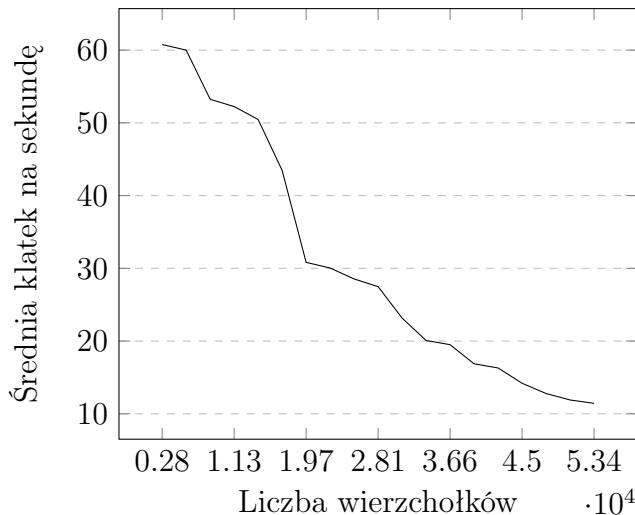
Jednakże, w trakcie testów wydajności wykryto błąd krytyczny, który nie był związany z funkcjonalnością, ale z działaniem programu i polegał na okresowych awariach.

Tryb debugowania został wykorzystany do próby odkrycia przyczyny problemu niespodziewanych awarii programu. Okazało się, że błąd objawiał się czasem w momencie, gdy OpenGL tworzy i inicjalizuje siatkę buforową. Nie było jednak wystarczających informacji o przyczynie awarii, aby naprawić problem.

3.1 Wydajność

Wydajność oprogramowania jest dość mocno uzależniona od używanego sprzętu. Na testowanym przeze mnie urządzeniu wydajność wzrostu drzew jest bezpośrednio związana z liczbą umieszczonych drzew, a także ich złożonością. Parametry drzewa, takie jak Levels i Tree steps, mają szczególnie duży wpływ na wydajność. Są to jedynie parametry, które zmieniają liczbę wierzchołków i krawędzi drzewa. Wszystkie inne parametry zmieniają jedynie położenie wierzchołków. Każde drzewo przy Levels i Tree steps ustawionych na 5 ma 2183 węzły i 6045 krawędzi. Jeśli jednak ustawimy oba te parametry na 6, to liczba węzłów wzrośnie do 5408 węzłów i 13806 krawędzi. Dlatego im więcej poziomów ma drzewo, tym większe będzie obciążenie urządzenia i wolniejsze generowanie drzewa.

Przetestowałem wydajność wzrostu drzew i zrobiłem wykres zależności średniej klatek na sekundę (ang. *FPS*) od liczby wierzchołków.



Rysunek 3.1: Wykres zależności średniej klatek na sekundę od liczby drzew

3.2 Porównanie z innymi rozwiązaniami

Istnieje jeszcze kilka innych rozwiązań, które generują trójwymiarowe drzewa. Oto następujące programy o podobnej funkcjonalności do mojego rozwiązania:

- Snappytree [8],
- PD: Tree Generator [9],
- HappyTree [6].

Snappy tree i PD: Tree Generator używa biblioteki proctree.js jako podstawy, podczas gdy Happy Tree używa portu biblioteki proctree, tak jak program, który stworzyłem. Charakterystyczną cechą mojego programu jest symulacja wzrostu drzew. Taka funkcjonalność nie jest dostępna w wymienionych programach. Przykład obecności lub braku danej cechy przedstawiono w tabeli 3.1.

Tabela 3.1: Porównanie możliwości rozwiązań

| | Moje rozwiązanie | PD: Tree Generator | Snappy Tree | Happy Tree |
|--|------------------|--------------------|-------------|------------------|
| Ustawienie parametrów generacji drzewa | Tak | Tak | Tak | Tak |
| Zapis parametrów do pliku JSON | Tak | Tak | Tak | Nie |
| Obsługa wielu drzew | Tak | Nie | Nie | Tak ¹ |
| Ładowanie tekstur nie-standardowych | Tak | Nie | Nie | Nie |
| Zapis modelu do pliku OBJ | Nie | Tak | Tak | Tak |
| Ustawienie oświetlenia | Nie | Tak | Nie | Nie |

1 Nie można wpływać na liczbę i lokalizację drzew

Wiele z alternatywnych rozwiązań zapewnia funkcjonalność, która nie została za-implementowana w moim rozwoju. Jednak moje rozwiązanie dostarcza wąsko ukie-

runkowaną funkcjonalność, która została przygotowana we współpracy z pracownikami Instytutu Nauk Biologicznych Uniwersytetu Marii Curie Skłodowskiej w Lublinie.

Podsumowanie

Celem niniejszej pracy było opracowanie oraz zastosowanie systemu Lindenmaera w symulacji wzrostu roślin.

System Lindenmaera jest zbiorem reguł, które pozwalają na zdefiniowanie generacji struktur fraktalnych. Pomimo swojej prostoty L-system posiada cechy rozszerzalności, dzięki czemu możliwe jest opisywanie złożonych struktur, w tym roślin świata rzeczywistego [10].

Na wstępie przedstawiono szczegółowe dotyczące systemu Lindenmeyera, przykłady zastosowania i wyniki w przestrzeni dwuwymiarowej oraz opis tworzenia drzew w przestrzeni trójwymiarowej. Następnie przedstawiono implementację oprogramowania pozwalającego na projektowanie lokalizacji drzew, możliwość ich parametryzacji oraz symulację zmian w czasie. Ostatnim krokiem było przetestowanie stworzonego oprogramowania i porównanie go z innymi podobnymi rozwiązaniami.

Wadą przedstawionego rozwiązania jest niestabilność programu, która w szczególnych przypadkach objawia się zatrzymaniem programu. Ten problem nie został rozwiązany. Nie ma też symulacji oświetlenia, co mogłoby być dobrym dodatkiem.

Bibliografia

- [1] M. Christenhusz and W. J. Byng, “The number of known plant species in the world and its annual increase,” 2016. https://www.researchgate.net/publication/303371386_The_number_of_known_plant_species_in_the_world_and_its_annual_increase, (dostęp 2022-06).
- [2] A. Lindenmayer, “Interactive modeling of plants,” *Journal of Theoretical Biology*, 1968.
- [3] R. Coelho, N. Calonego, and L. Consularo, “Visualization and simulation of 3d artificial neural structures generated by l-system,” *Virtual Reality*, 2010.
- [4] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.
- [5] P. Brunt, “proctree.js,” 2012. <https://github.com/supereggbert/proctree.js>, (dostęp 2022-06).
- [6] J. Komppa, “proctree,” 2015. <https://github.com/jarikomppa/proctree>, (dostęp 2022-06).
- [7] N. Lohmann, “Nlohmann json,” 2015. <https://github.com/nlohmann/json>, (dostęp 2022-06).
- [8] P. Brunt, “Snappy tree,” 2012. <http://snappytree.com/>, (dostęp 2022-06).
- [9] A. Marsh, “Pd: Tree generator,” 2010. <https://drajmarsh.bitbucket.io/tree3d.html>, (dostęp 2022-06).

- [10] G. Rozenberg and A. Salomaa, *Lindenmayer Systems. Impacts on Theoretical Computer Science, Computer Graphics, and Development Biology*. Springer-Verlag, Berlin, 1992.