|  |
| --- |
| Projekt nr 25: Generowanie i animacja fraktali |
| Jan Paluch, Maciej Mikołajek, Wiktoria Szewczyk |

# Opis projektu

Głównym celem projektu jest napisanie programu, który wyświetla dwuwymiarowe fraktale. Dodatkowo tworzy animację przekształcając („morfując”) różne fraktale między sobą. Program umożliwia wyświetlanie dowolnych fraktali, jeśli dostarczymy mu dane na temat jego przekształceń afinicznych.

# Założenia wstępne przyjęte w realizacji projektu

Jednym z podstawowych założeń projektu jest wczytywanie z pliku parametrów animacji, między innymi rozmiar animacji, liczba klatek oraz liczba fraktali oraz ich parametry (odpowiednie współczynniki przekształceń afinicznych).

Program wyświetla na ekranie fraktale na podstawie dostarczonych danych.

Nasz program powinien sam zadbać o odpowiednie skalowanie rozmiaru fraktali, tak aby mieściły się na ekranie i nie były za małe. Dodatkowo wyświetlane fraktale dostosowują się dynamicznie do zmiany rozmiaru okna.

Kolor pikseli wyświetlanego fraktala powinien zależeć od tego, które przekształcenie zostało użyte do obliczenia jego pozycji.

# Analiza projektu

**Specyfikacja danych wejściowych**

Rozmiar animacji: rozmiar x i y bitmap.

Ilość iteracji: ilość punktów, z których złożony jest fraktal.

Ilość fraktali.

Ilość przekształceń afinicznych dla konkretnego fraktala.

Przekształcenie: współczynniki A, B, C, D, E, F przekształcenia afinicznego.

**Oczekiwane dane wyjściowe**

Na ekranie wyświetla się wybrany fraktal, zbudowany z odpowiednio obliczonych punktów, o odpowiednich kolorach. Dodatkowo po kliknięciu w przycisku „Morfuj” widzimy animacje przejścia między kolejnymi fraktalami (jeśli na wejściu podaliśmy dane dla więcej niż jednego fraktala).

**Struktury danych**

Zdecydowaliśmy się na stworzenie 4 klas: *Point*, *Trans*, *TransSet* oraz *Instruction*.

Klasa *Point* przechowuje przechowuję punkt w układzie kartezjańskim, razem z jego kolorem wyświetlania.

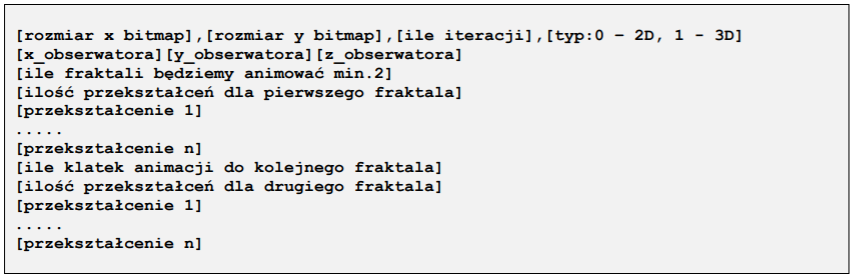
Klasa *Trans* przechowuję dane na temat jednej transformacji (współczynniki A, B, C, D, E, F) oraz kolor, który będą miały punkty stworzone za jej pomocą. Zajmuje się też obliczaniem punktu na podstawie poprzedniego.

Klasa *TransSet* przechowuję wszystkie transformacje wymagane do wygenerowania jednego fraktala oraz ich ilość. Ponadto zajmuje się wyborem losowej transformacji.

Klasa *Instruction* przechowuje instrukcje generujące fraktale.

**Specyfikacja interfejsu użytkownika**

Użytkownik może wprowadzić dane za pomocą wcześniej przygotowanego pliku, którego struktura jest następująca:



Zakładamy, że tworzymy tylko fraktale 2D więc dostarczone dane dotyczące pozycji obserwatora są pomijane przez program. Przykładowy plik został dołączony do katalogu BIN.

Użytkownik może za pomocą przycisku „Następny” wybierać, który fraktal chce wyświetlić lub wyświetlić wszystkie w po kolei w postaci animacji klikając przycisk „Morfuj”.

**Wyodrębnienie i zdefiniowanie zadań**

Projekt podzieliliśmy na następujące zadania, które kolejno wykonywaliśmy:

* Opracowanie planu struktury programu
* Zbudowanie interfejsu
* Wczytywanie danych z pliku
* Implementacja struktur danych
* Wyświetlanie pojedynczego fraktala
* Animacja przejść między fraktalami
* Skalowanie fraktali

**Wybór narzędzi programistycznych**

Wybraliśmy środowisko Microsoft Visual Studio ze względu na, że jesteśmy z nim już zaznajomieni oraz na wysokiej jakości narzędzia, które nam ono oferuje.

Używaliśmy języka C++ w standardzie C++14.

Dodatkowo korzystaliśmy z biblioteki wxWidgets, ponieważ umożliwiła nam ona łatwe zaimplementowanie interfejsu oraz wyświetlanie obrazów.

# Podział pracy i analiza czasowa

Regularnie spotykaliśmy się, żeby pracować nad projektem razem. Na początku zapoznaliśmy się z celami projektu i opracowaliśmy plan na struktury danych w projekcie. Praca nad projektem została rozłożona równomiernie. Aspekty wczytywania danych z pliku, tworzenia struktur danych lub proces wyświetlania fraktali na ekranie mogliśmy rozdzielić między siebie, jednak trudniejsze zagadnienia jak obliczanie i skalowanie kolejnych punktów lub sposób animacji omawialiśmy i implementowaliśmy wspólnie.

# Opracowanie i opis niezbędnych algorytmów

Przekształcenie afiniczne ma postać:

Pierwszy punkt zawsze jest punktem , a następne obliczane są na podstawie poprzedniego. Parametry do przekształcenia są dla każdego fraktala losowane z odpowiadającego mu zbioru przekształceń.

Parametry te, dostarczane są poprzez plik tekstowy. Plik ten następnie jest deserializowany do struktur danych, dla prostoty wykonywania procedur.

Skalowanie punków fraktala odbywa się za pomocą wbudowanej w wxWidgets procedury skalowania bufora.

Morfizm między dwoma fraktalami zaimplementowany jest jako liniowa transformacja odpowiadających sobie punktów w wyliczonych fraktalach.

# Kodowanie

W programie po uruchomieniu widoczne są 3 przyciski, pozwalające kolejno na wczytanie pliku sterującego, morfowanie między fraktalami oraz bezpośrednie przechodzenie przez fraktale.

Funkcja pierwsza wczytuje i odczytuje plik, deserializując go do obiektów z napisanych struktur, następnie wyświetla w panelu pierwszy fraktal i informuje użytkownika, że program jest gotowy do kontynuowania pracy. Wszystkie położenia punktów są już wyliczone, zatem następne zadania powinny wykonywać się stosunkowo szybko i płynnie.

Następnie użytkownik może wyświetlić kolejny fraktal przechodząc do niego bezpośrednio, bądź animując przejście (morfując). Morfizm transformuje po linii każdy punkt pierwotnego fraktala w następujący po nim, natomiast kliknięcie przycisku „Następny” po prostu rysuje kolejny fraktal z listy.

# Testowanie

Testowaliśmy projekt na podstawie przykładowej instrukcji sterującej, zmieniając jej parametry, tworząc własne procedury bądź usuwając je. Ponieważ obrany przez nas proces kodowania pozwalał na szybką iteracje, udawało nam się sprawnie wyłapywać wszystkie błędy. Korzystanie z systemu kontroli wersji GIT pozwoliło na asynchroniczną pracę nad wieloma funkcjami jednocześnie. Przetestowaliśmy różne scenariusze – gdy fraktali mamy 2, 5 czy 1. Co się stanie, gdy mamy 10 000 punktów, a co gdy mamy 50 000 czy 1 000 000. Jak wpłynie rozdzielczość na wygląd, ilość punktów na czas pracy czy zajmowaną pamięć.

Złożoność wyświetlania jednej klatki można oszacować na O(n), gdzie n to ilość punktów, podobnie w przypadku zajmowanego miejsca przez tablice punktów. Czas morfizmu zależny jest od ilości punktów, klatek przy przejściu oraz ilości fraktali. Najlepsze rezultaty (płynność i jakość) uzyskaliśmy dla 30 000 punktów.

Program testowaliśmy na komputerach z systemem Windows, architekturze i86\_64, przy rozdzielczościach ekranu Full HD oraz 4K.

# Wdrożenie, raport i wnioski

Można stwierdzić, że program wdrożono i wykonano z powodzeniem, wszystko wykonuje się zgodnie z oczekiwaniami.

W kolejnej wersji zadbalibyśmy o obsługę wyjątków – co się stanie jeśli plik wejściowy nie będzie poprawny do utworzenia struktur? Co jeśli zabraknie miejsca w pamięci bądź czas obliczeń będzie zbyt długi aby miał sens?

Następnie przepisalibyśmy funkcje czytającą plik oraz morfującą tak, aby działały one asynchronicznie.

Kolejnym krokiem mogłoby być dodanie elementów interfejsu użytkownika pozwalających na zmianę ilości punktów bądź przybliżanie czy przesuwanie fraktala już po wczytaniu procedur.

Sądzę iż powyższe zmiany, w szczególności asynchronizacja, podniosłaby użyteczność projektu, lecz nie jest to zadanie trywialne.