UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Adrian Rupala

Krzywe generowane fraktalnie

(projekt zaliczeniowy z elementów animacji i grafiki 3D)

Sosnowiec 2019

1. Wstęp

Fraktale to zbiory o skomplikowanej, rekurencyjnej budowie. Kryją w sobie nieskończone samopodobieństwo. Można je dostrzec w przyrodzie, między innymi w budowie kalafiora czy rozgałęzień naczyń krwionośnych. W grafice komputerowej fraktale wykorzystuje się do tworzenia różnorodnych losowych krajobrazów lub map geograficznych.

Głównym celem projektu było stworzenie prostej aplikacji terminalowej dla systemu Ubuntu GNU/Linux generującej popularne krzywe (Krzywą Kocha, Smoka Heighwaya, Krzywą Hilberta oraz Trójkąt Sierpińskiego), które można uznać za obiekty fraktalne ze względu na ich rekurencyjną definicję oraz samopodobny układ.

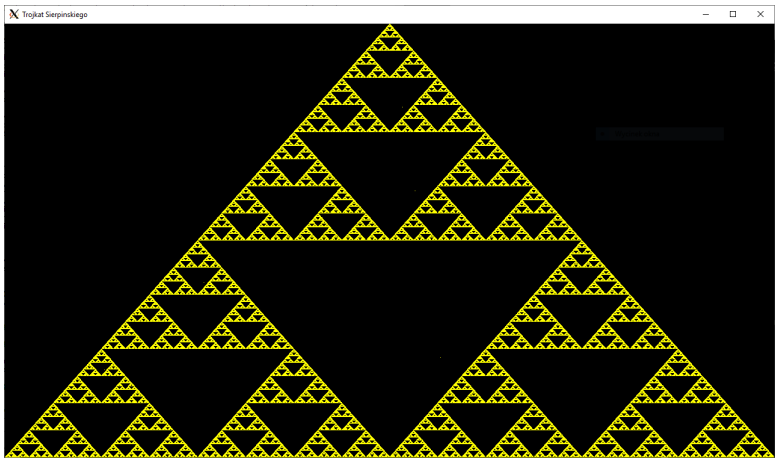
Użytkownik po uruchomieniu aplikacji wybiera z prostego, kontekstowego menu obiekt, który ma zostać narysowany, a następnie musi podać maksymalną głębokość (ilość generowanych powtórzeń danego obiektu).

2. Teoria i algorytmy

Istnieje wiele możliwości wygenerowania krzywych fraktalnych. Każdy z wygenerowanych w aplikacji obrazów jest krzywą o odpowiednich właściwościach.

Trójkąt Sierpińskiego (Rysunek 2.1) został wygenerowany za pomocą metody "Gry w chaos". Jest to algorytm komputerowego generowania fraktali, w którym wygenerowany zostaje przybliżony obraz atraktora (zbioru w przestrzeni, do którego w miarę upływu czasu zmierzają wszystkie trajektorie na danym obszarze) lub punktu stałego dowolnego systemu funkcji iterowanych. Algorytm zaczyna od pewnego punktu x_0 następnie każda kolejna iteracja odbywa się za pomocą wzoru $x_{n-1} = f^m(x_n)$, gdzie $f^m(x)$ jest jedną z funkcji iterowanych, losowo wybieraną dla każdej iteracji. Gdy wartość początkowa x_0 należy do atraktora systemu funkcji iterowanych, wówczas wszystkie punkty x_n również do niego należą i z prawdopodobieństwem 1 tworzą zbiór gęsty.

W przypadku wygenerowania Trójkąta Sierpińskiego należy przyjąć na początku postawienie 3 dowolnych punktów, po czym algorytm wybiera sobie kolejny punkt płaszczyzny. Następnie wybrany zostaje dowolny z początkowych punktów i stawia się punkt w połowie odległości między czwartym punktem a wybranym. Powtarza się ten krok, za każdym razem oznaczając punkt leżący dokładnie w połowie odległości między nowym punktem a jednym z trzema pierwszych. Efektem algorytmu jest jeden z wariantów Trójkąta Sierpińskiego. Wierzchołki to ustalone na początku punkty.



Rysunek 2.1. Trójkąt Sierpińskiego dla 100 * 10000 punktów

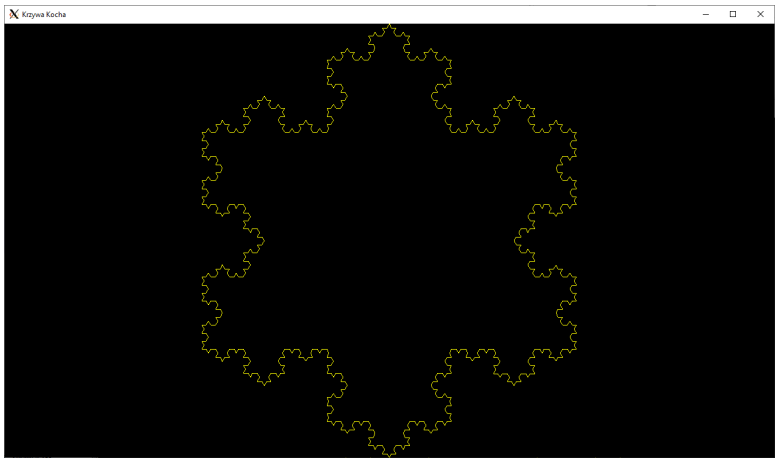
Trzy pozostałe krzywe - Krzywa Kocha, Smok Heighwaya oraz Krzywa Hilberta zostały wygenerowane za pomocą podążania ścieżką wzdłuż krzywej, a następnie zmieniając krzywą w miarę wzrostu głębokości fraktala.

W przypadku generowania Krzywej Kocha (Koch Snowflake) produkt końcowy powstaje z odcinka, który następnie jest podzielony na 3 części i jego środkowy podział zostaje zastąpiony ząbkiemó ramieniu długości $\frac{1}{3}$ odcinka takim, że wraz z usuwaną częścią zostaje utworzony trójkąt równoboczny. Krok jest powtarzany tyle razy, ile poda użytkownik.

— Krok 0. Krzywa Kocha w kroku zerowym $k = 0$ jest odcinkiem. Zostaje on podzielony na 3 równe części, a środkową zastąpią dwa odcinki długości $\frac{1}{3}l$, nachylone względem niej pod kątem 60°. Wraz z wyciętym fragmentem mogłyby one utworzyć trójkąt równoboczny.

— Krok 1. Krzywa Kocha w kroku pierwszym $k = 1$, po transformacji zawiera 4 odcinki, każdy równy $\frac{1}{3}l$. W kolejnym kroku każdy z tych odcinków ponownie zostanie podzielony na 3 części, a środkową znów zastąpimy dwoma odcinkami.

— Krok 2. Krzywa Kocha w kroku drugim $k = 2$ zawiera już 16 odcinków, każdy długości $\frac{1}{9}l$. W kolejnym kroku $k = 3$ powstanie ich 64 (Rysunek 2.2), każdy długości $\frac{1}{27}l$.



Rysunek 2.2. Krzywa Kocha, podział k = 3

Smok Heinghwaya (Rysunek 2.3) to przykład kolejnej krzywej, jaką jest w stanie wygenerować aplikacja. Może być on zdefiniowany jako atraktor systemu funkcji zwężających zapisanego w notacji zespolonej dla zestawu dwóch punktów $S_0 = \{0, 1\}$.

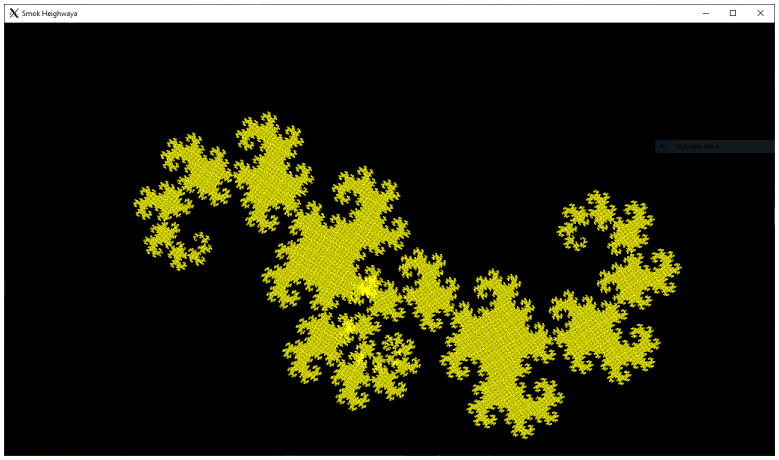
$$f_1(z) = \frac{(1+i)z}{2}$$

$$f_2(z) = \frac{(1-i)z}{2}$$

Najczęściej jednak w oprogramowaniu, również w przypadku tej aplikacji zastosowane są przekształcenia bazujące na funkach posiadające liczby rzeczywiste:

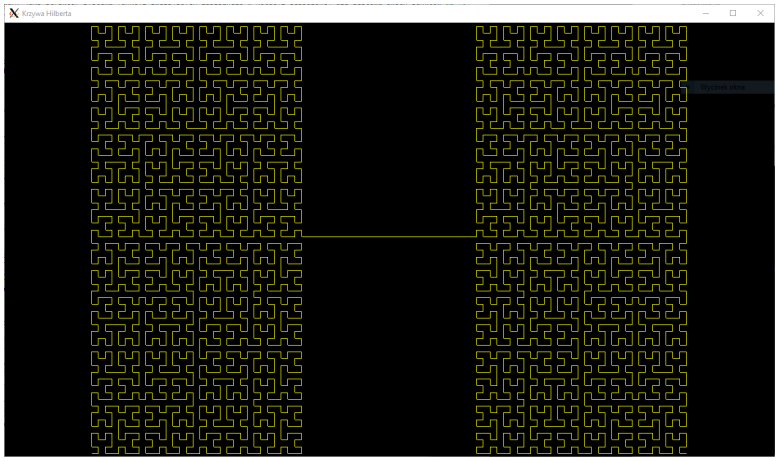
$$f_1(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \cos 45 & -\sin 45 \\ \sin 45 & \cos 45 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_2(x,y) = \frac{1}{\sqrt{2}} \begin{pmatrix} \cos 135 & -\sin 135 \\ \sin 135 & \cos 135 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$



Rysunek 2.3. Smok Heighway dla 16 iteracji

Ostatnią krzywą, która została zaprogramowana w ramach tego projektu, jest Krzywa Hilberta (Rysunek 2.4). Jest to krzywa, która wypełnia płaszczyznę, przechodząc przez jej wszystkie punkty. Krzywa ta powstaje w wyniku połączenia łamania kwadratów powstałych z podziału kwadratu podstawowego kolejno na 4, 16, 64 itd. Zarówno krzywa Hilberta, jak i jej przybliżenia są używane w informatyce, między innymi w adresacji IP, ponieważ dają odwzorowanie między przestrzenią 1D a 2D, która dość dobrze zachowuje lokalizację. Jeśli punkty (x, y) są współrzędnymi punktu w obrębie kwadratu jednostki, a d jest odległością wzdłuż krzywej, gdy osiągnie ten punkt, to punkty, które mają pobliskie wartości d będą miały również wartości pobliskie (x, y). Nawet gdy punkty w obrębie współrzędnych $(x.y)$ będą znajdowały się blisko siebie, ich wartość d będzie oddalona.



Rysunek 2.4. Krzywa Hilberta dla 5 iteracji

3. Opis programu

Program został napisany w języku programowania C++ z wykorzystaniem technologii OpenGL oraz biblioteki graficznej GLFW. Do skompilowania programu został wykorzystany projekt GNU Compiler Collection a konkretnie jego część będąca kompilatorem g++. W celu ułatwienia kompilacji został utworzony plik "makefile"będący zalecanym sposobem kompilacji projektu.

Aby ułatwić przygotowanie potrzebnych bibliotek dla systemów opartych na dystrybucji Debian GNU/Linux w tym Ubuntu GNU/Linux został również przygotowany prosty skrypt instalacyjny, który to po wywołaniu instaluje wszystkie potrzebne biblioteki.

Do edycji pliku źródłowego został wykorzystany edytor Visual Studio Code wraz z następującymi dodatkami: Microsoft C/C++ oraz C++ Intellisense.

3.1. Możliwości programu

Program po uruchomieniu potrafi narysować wybraną przez użytkownika końcowego krzywą zgodnie z podaną przez niego liczbą zagnieżdżeń. Program potrafi narysować następujące krzywe:

— Krzywą Kocha (Koch snowflake)
— Smoka Heighwaya (Dragon curve)
— Krzywą Hilberta (Hilbert curve)
— Trójkąt Sierpińskiego (Sierpinski gasket)

3.2. Opis programu

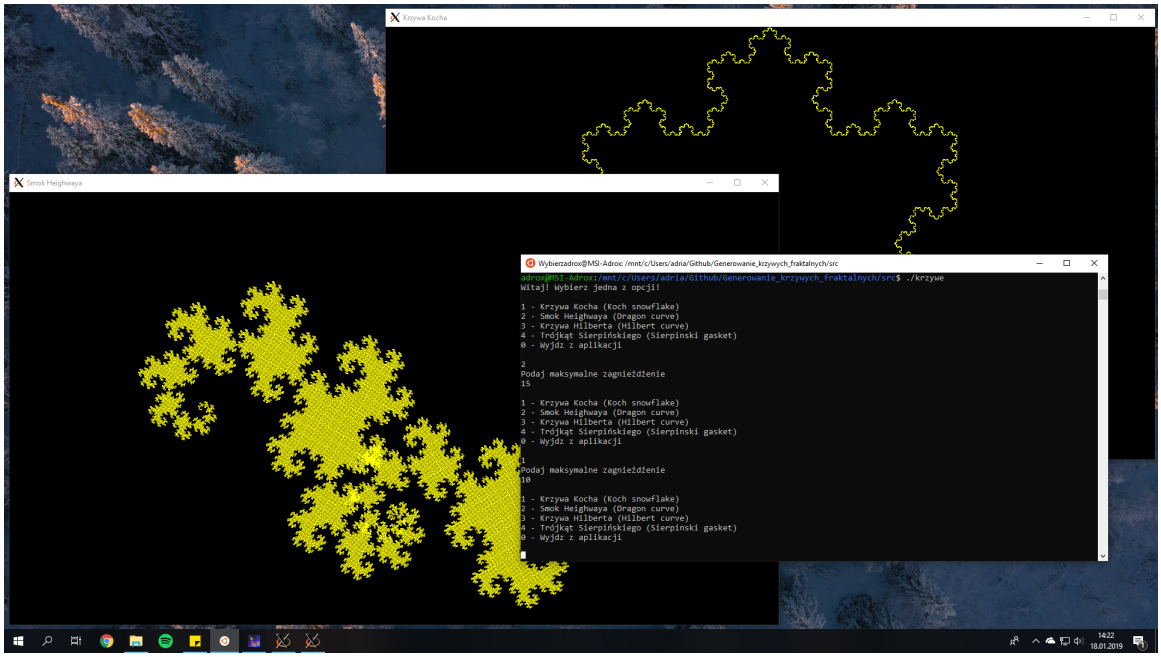
Aplikacja została przygotowana z myślą o wywołaniu jej z poziomu powłoki Bash. W celu uruchomienia aplikacji należy uruchomić konsolę, przejść do folderu z aplikacją za pomocą polecenia `$ cd sciezka/do/folderu`. Gdy znajdujemy się w katalogu programu, aby uruchomić aplikację, należy użyć polecenia `$./krzywe`.

Aby skompilować aplikację, należy znajdować się w folderze z plikiem źródłowym `main.cpp` oraz plikiem make `makefile` a następnie wpisać komendę `$ make`.

Aby uruchomić plik postinstalacyjny mający na celu przygotować wszystkie potrzebne biblioteki należy przejść do katalogu `scripts`, nadać uprawnienia do wykonywania pliku za pomocą polecenia `$ chmod +x post.sh` a następnie wykonać skrypt za pomocą komendy `./post.sh`. Skrypt do działania wymaga podania hasła administratora ze względu na instalację bibliotek z różnych repozytoriów.

Istnieje możliwość uruchomienia aplikacji w środowisku Windows Subsystem for Linux (dla podsystemu Ubuntu) (Rysunek 3.1). W tym celu należy zainstalować dodatkowo aplikację `vcxsrv` emulującą X Server dla systemów Windows oraz po uruchomieniu podsystemu należy wykonać komendę `$ export DISPLAY=localhost:0.0`, która to umożliwia wyexportowanie wyświetlanego obrazu na zewnętrzny system (Windows 10). Należy jednak pamiętać, aby wcześniej uruchomić aplikację `vcxsrv` z następującymi parametrami:

— Multiple windows
— Start no client
— Clipboard, Primary selection (odznaczyć Native opengl)



Rysunek 3.1. Aplikacja uruchomiona na systemie Windows 10

Bibliografia

[1] Dokumentacja biblioteki GLFW
https://www.glfw.org/documentation.html

[2] Dokumentacja OpenGL API
http://docs.gl/

[3] Dokumentacja języka C++
https://devdocs.io/cpp/

[4] Metodologia Chaos Game
https://rosettacode.org/wiki/Chaos_game

[5] Gra w chaos - Wikipedia
https://pl.wikipedia.org/wiki/Gra_w_chaos

[6] Krzywa Kocha - Wikipedia
https://pl.wikipedia.org/wiki/Krzywa_Kocha

[7] Smok Heighwaya - Wikipedia
https://pl.wikipedia.org/wiki/Smok_Heighwaya

[8] Krzywa Hilberta - Wikipedia
https://pl.wikipedia.org/wiki/Krzywa_Hilberta

[9] Goldman, R.: The Fractal Nature of Bézier Curves. Proceedings of Geometric Modeling and Processing: Theory and Applications, pp. 3-11, (2004)

[10] Kotarski, W.: Fraktalne modelowanie kształtu. EXIT, (2008)

[11] Schaefer, S., Levin, D., Goldman, R.: Subdivision Schemes and Attractors. Eurographics Symposium on Geometry Processing, pp. 171-180, (2005)