Rysowanie transformatą Fouriera

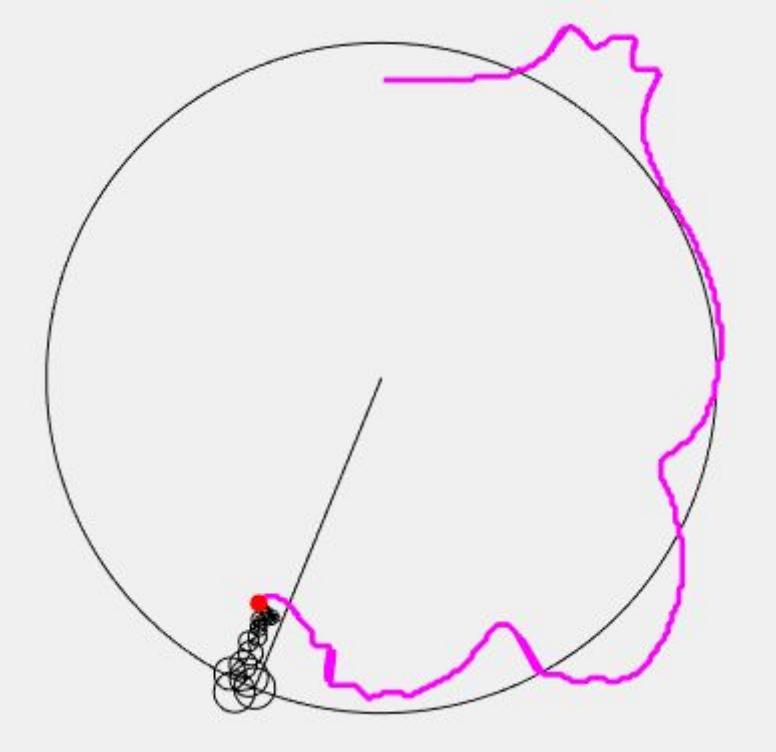
DOKUMENTACJA PROJEKTU

Analiza Obrazów 2020/21

Piotr Harmuszkiewicz, Tomasz Gajda

I. Opis projektu

Tematem projektu jest aplikacja pozwalająca użytkownikowi na odtworzenie animacji rysowania kształtów korzystając z epicykli Fouriera.



Rysunek 1: Przykład obrazu rysowanego przez epicykle Fouriera

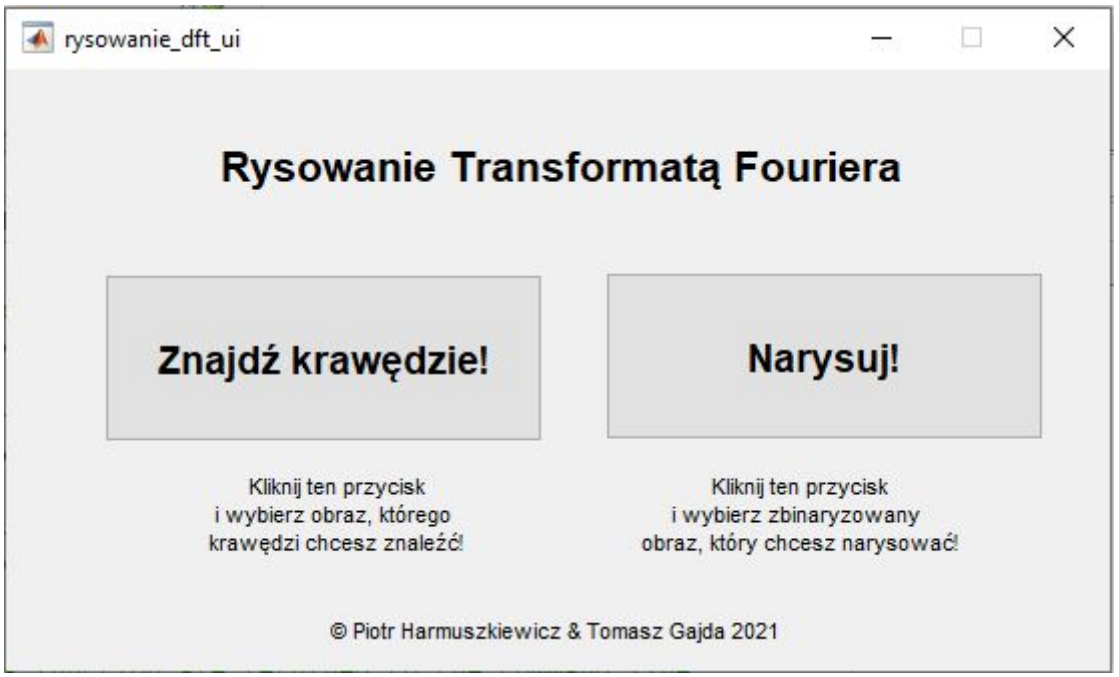
Użytkownik po wprowadzeniu danych obrazu potrzebnych do stworzenia takiej animacji, powinien zobaczyć cały proces rysowania krawędzi obrazu.

Dodatkowym wymaganiem jest możliwość przetworzenia obrazu, na dane opisujące jego krawędzi, które mogłyby zostać wykorzystane w późniejszym rysowaniu.

II. Interfejs

Interfejs naszego programu został stworzony za pomocą **GUIDE Layout Editor**. Jest on bardzo prosty, ponieważ składa się jedynie z dwóch **przycisków** i kilku statycznych **napisów**. Cały kod, na którym opiera się interfejs można znaleźć w pliku ***"rysowanie_dft_ui.m"***.

Sprowadza się on do nadania funkcji jako odpowiedzi na kliknięcie obu z przycisków.



Rysunek 2: Interfejs programu

Jeden z nich odpowiada za przygotowanie obrazu do rysowania, drugi natomiast odtwarza animację rysowania wcześniej spreparowanego wybranego zestawu danych.

III. Poradnik użytkownika

Do uruchomienia projektu potrzebne są:

- **MATLAB R2020b** lub nowszy
- pobrany katalog z projektem

By włączyć projekt trzeba uruchomić plik ***"rysowanie_dft_ui.m"*** czyli plik z interfejsem programu. Po wyświetleniu interfejsu, mamy dwie opcje:

1. Jeśli chcemy przygotować obraz do rysowania, klikamy przycisk **"Znajdź krawędzie!"**, wybieramy obraz, którego krawędzie chcemy odnaleźć, a następnie zapisujemy dane do rysowania w formacie **.mat** (zalecany zapis do folderu ImageData) oraz sam rysunek krawędzi w formacie **.bmp** (zalecany zapis do folderu ImageEdge).

2. Gdy mamy już dane gotowe do rysowania, klikamy przycisk **"Narysuj!"**, który pozwala nam wybrać plik z danymi w formacie **.mat** i odtwarza animację rysowania.

IV. Działanie

Przygotowanie obrazka:

Rozpoczynamy od **zbinaryzowania** obrazu.

1. Binaryzacja musi być **dokładna**, więc zalecamy ręczne dobranie progu binaryzacji lub użycie obrazu z jednolitym białym tłem. Binaryzacja nie obsługuje wielu obiektów na obrazie, jednak można wyciągnąć pojedynczy obiekt, przy użyciu funkcji nadającej etykiety.

2. Następnie wypełnimy wszystkie dziury w środku naszego obiektu i wykonujemy **otwarcie** i **zamknięcie**, aby pozbyć się pojedynczych pikseli.

3. Kolejnym krokiem jest użycie funkcji ***edge()***, która zwróci nam tylko krawędzie obrazu oraz obliczenie centrum masy naszego obiektu (centrum masy w tym przypadku będzie średnią arytmetyczną współrzędnych x i y).

4. Następnie dla każdego piksela na krawędzi, zamieniamy jego współrzędne na współrzędne biegunowe ze środkiem masy jako punkt **(0, 0)**. Dzięki temu możemy w łatwy sposób ustawić nasze punkty w kolejności rysowania - wystarczy, że posortujemy po **thecie**. Dla niektórych obrazów będzie to jednak powodowało problemy. Gdy dwa kąty są zbliżone, a między punktami znajduje się przerwa (np. dla wiewiórki - pomiędzy plecami a ogonem znajduje się przerwa) algorytm nie zadziała poprawnie, co zauważymy przy rysowaniu.

5. Na końcu wracamy współrzędnymi do współrzędnych kartezjańskich i zapisujemy je do dwóch wektorów **X** i **Y**.

Transformata Fouriera:

1. Zaczynamy od obliczenia **dyskretnej transformaty Fouriera**, aby to zrobić, musimy zapisać nasze współrzędne jako liczby zespolone. Z transformaty otrzymujemy: **częstotliwość**, **promień** oraz **fazę**.

2. Następnie wszystkie te elementy sortujemy po **promieniu**, ponieważ podczas rysowania chcemy rozpoczynać od **największego** koła i przechodzić po coraz **mniejszych** kołach.

Rysowanie:

1. Ustalamy krok czasu, o jaki będziemy się przesuwać z **każdą** iteracją.

2. Tworzymy **tablicę**, w której będziemy przechowywać wszystkie elementy naszej krzywej, aby nie utracić zawartości przy przerysowywaniu, gdyż musimy czyścić **„płótno“**.

3. Zaczynamy od obliczenia środków każdego koła i zapisujemy je w naszej tablicy. Zapisujemy również **linie** (od środka poprzedniego koła do środka aktualnego koła), które będą promieniem koła, co poprawi **czytelność** animacji.

4. Następnie za pomocą funkcji **viscircles()** rysujemy wszystkie koła i używamy funkcji **„hold on“**, aby przy rysowaniu promieni i krzywej, płótno **nie zostało wyczyszczone**.

5. Kolejnym krokiem jest narysowanie **promieni** dla każdego koła, a następnie samej **krzywej**.

6. Na końcu zaznaczamy **punkt**, w którym aktualnie rysujemy, za pomocą **czerwonej kropki**.

V. Co nie działa?

1. Najpoważniejszy błąd naszej aplikacji występuje, gdy promień wypuszczony z punktu (0, 0), czyli środka masy, pod pewnym kątem theta trafia w 2 lub więcej pikseli, jest to zademonstrowane na pokemonie **Squirtle**, a dokładnie na jego ogonie.

   Problem polega na tym, że nie jesteśmy w stanie cofnąć się naszym promieniem, aby podążać za konturem, a jedna wartość theta odpowiada jednemu pikselowi.

   Z tego powodu, podczas sortowania, ustalamy niepoprawną kolejność pikseli, co powoduję przeskoki między dwoma niepołączonymi pikselami.

   Rozwiązaniem tego problemu jest podzielenie obrazu na dwie oddzielne funkcje: funkcje reprezentujące współrzędne x oraz y i wyznaczenie transformaty Fouriera, a następnie narysowanie konturów na podstawie 2 transformat.

2. Jeśli wyłączymy animację rysowania przed jej zakończeniem, Matlab wyrzuca **błąd**. Jest to element uciążliwy dla użytkownika i wymaga poprawienia.

3. Binaryzacja działa poprawnie jedynie na **szczególnych** obrazkach: muszą one posiadać białe tło, choć zdarza się że nawet na takich czasem zdarza się jakiś błąd, którego nie potrafimy rozszyfrować

VI. Podział pracy

Piotr Harmuszkiewicz:

- funkcja rysująca,
- funkcja binaryzująca,
- opis działania (i niedziałania)

Tomasz Gajda:

- interfejs, połączenie z funkcjami,
- pomniejsze dodatkowe funkcje,
- dokumentacja

VII. Źródła

- https://en.wikipedia.org/wiki/Discrete_Fourier_transform - strona na wikipedii na temat dyskretnej transformaty Fouriera
- https://www.myfourierepicycles.com/ - strona dobrze opisująca zjawisko
- https://www.mathworks.com/matlabcentral/fileexchange/72821-drawing-with-fourier-epicycles - podobna aplikacja stworzona przez **Víctor Martínez-Cagigal**
- https://www.youtube.com/watch?v=MY4luNgGfms - film z kanału **The Coding Train**, dobrze przedstawiający działanie w kodzie