

UNIWERSYTET GDAŃSKI
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI

Marcin Belicki
numer albumu: 273417

Kierunek studiów: Informatyka

OTOCZKI WYPUKŁE

Praca magisterska
wykonana
pod kierunkiem
dr inż. Arkadiusz Mirakowski

Gdańsk 2023

Spis treści

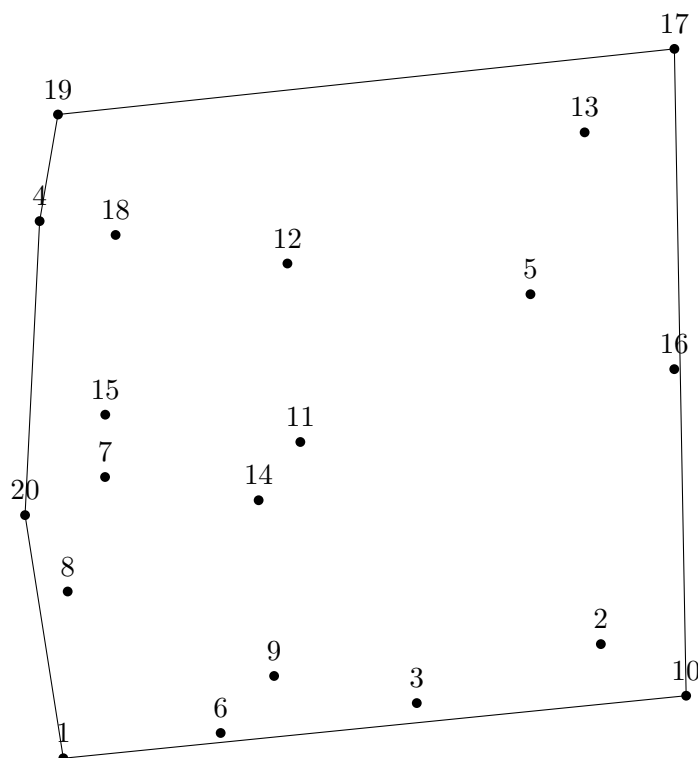
Wstęp	3
Rozdział I Otoczka wypukła na płaszczyźnie	4
1. Otoczka wypukła zbioru punktów	4
2. Otoczka wypukła wielokąta prostego	13
3. Redukcja zbioru punktów do wielokąta prostego	13
Rozdział II Zastosowania	14
1. Generalizacja kartograficzna	14
2. Grafika komputerowa	15
3. Detekcja obiektów	16
4. Wyznaczanie obwiedni sygnału	16
Rozdział III Implementacja w języku Scala	18
1. Pojęcia ogólne	18
2. Algorytm Grahama	27
3. Algorytm Jarvisa	31
Rozdział IV Dynamiczna otoczka wypukła	33
1. Algorytm	33
2. Implementacja w języku Scala	33
Rozdział V Podsumowanie	34
Bibliografia	35

Wstep

Rozdział I

Otoczka wypukła na płaszczyźnie

Otoczka wypukła zbioru punktów w swojej najbardziej podstawowej postaci jest wielokątem wypukłym obejmującym wszystkie punkty ze zbioru punktów leżących na płaszczyźnie w taki sposób, aby wielokąt ten miał jak najmniejsze pole.



Rysunek 1: Otoczka wypukła na płaszczyźnie
Źródło: opracowanie własne

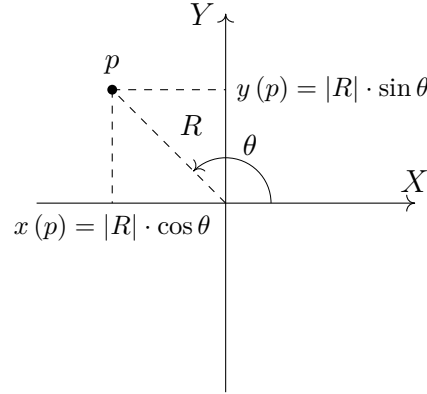
Dobłą reprezentacją otoczki wypukłej w świecie fizycznym może być grupa gwoździ przybita do płaskiej powierzchni i następnie opleciona ciasno sznurkiem. Gwoździe stykające się ze sznurkiem stanowią wierzchołki otoczki wypukłej tej grupy gwoździ.

1. Otoczka wypukła zbioru punktów

W celu wyznaczenia otoczki wypukłej dla najbardziej ogólnego przypadku — nieuporządkowanego zbioru punktów na płaszczyźnie, możemy wykorzystać dwa najbardziej popularne algorytmy algorytm Jarvisa oraz algorytm Grahama.

W wykorzystywanych algorytmach istotnym elementem jest sortowanie punk-

tów względem wartości, które zostały przedstawione na rysunku 2. W zależności od tego, w jakiej postaci określone będą dane punkty, należy dokonać odpowiednich obliczeń.



Rysunek 2: Punkt na układzie współrzędnych

Źródło: opracowanie własne

Gdzie poszczególne symbole oznaczają:

- p - rozpatrywany punkt
- $x(p)$ - odcięta punktu p
- $y(p)$ - rzędna punktu p
- R - długość wektora wodzącego punktu p
- θ - kąt nachylenia wektora wodzącego punktu p do osi OX

Istotną komplikację z punktu widzenia obliczeń w algorytmach może stanowić wyznaczenie wartości kąta θ (ze względu na potrzebę wykorzystania funkcji trygonometrycznych), którego dokładną wartość można wyliczyć za pomocą wzoru 1.

$$\theta(p) = \begin{cases} \arctg \frac{y(p)}{x(p)} & \text{dla } x(p) > 0; \\ \arctg \frac{y(p)}{x(p)} + \pi & \text{dla } x(p) < 0 \quad \wedge \quad y(p) \geq 0; \\ \arctg \frac{y(p)}{x(p)} - \pi & \text{dla } x(p) < 0 \quad \wedge \quad y(p) < 0; \\ \frac{\pi}{2} & \text{dla } x(p) = 0 \quad \wedge \quad y(p) < 0; \\ -\frac{\pi}{2} & \text{dla } x(p) = 0 \quad \wedge \quad y(p) > 0. \end{cases} \quad (1)$$

Należy jednak zauważyć, że do celów sortowania wystarczy zastosować funkcję $\alpha(p)$, taką, że dla dowolnej pary punktów p_1, p_2 spełnione będą warunki 2 oraz 3.

$$\alpha(p_1) < \alpha(p_2) \Leftrightarrow \theta(p_1) < \theta(p_2) \quad (2)$$

$$\alpha(p_1) = \alpha(p_2) \Leftrightarrow \theta(p_1) = \theta(p_2) \quad (3)$$

Przykładowa funkcja $\alpha(p)$ zachowująca spełniającą warunki 2 i 3 została przedstawiona za pomocą wzoru 4.

$$\alpha(p) = \begin{cases} \frac{y(p)}{d(p)} & \text{dla } x(p) \geq 0 \wedge y(p) \geq 0; \\ 2 - \frac{y(p)}{d(p)} & \text{dla } x(p) < 0 \wedge y(p) \geq 0; \\ 2 + \frac{|y(p)|}{d(p)} & \text{dla } x(p) < 0 \wedge y(p) < 0; \\ 4 - \frac{|y(p)|}{d(p)} & \text{dla } x(p) \geq 0 \wedge y(p) < 0. \end{cases} \quad (4)$$

Gdzie funkcja $d(p)$ określona jest zgodnie ze wzorem 5.

$$d(p) = |x(p)| + |y(p)| \quad (5)$$

W celu udowodnienia, że funkcja $\alpha(p)$ spełnia warunki 2 i 3, musimy wykazać, że funkcja $\gamma(\theta)$ (przedstawiona we wzorze 6), która stanowi przekształcenie funkcji $\alpha(p)$ takie, że jest w pełni zależna od wartości θ , a wartości $x(p(\theta))$ oraz $y(p(\theta))$ są określone zgodnie ze wzorem 7, jest rosnąca w każdym swoim przedziale.

$$\gamma(\theta) = \alpha(p(\theta)) \quad (6)$$

$$p(\theta) = (|R| \cdot \cos \theta, |R| \cdot \sin \theta) \quad (7)$$

Należy zauważyć, że poszczególne przypadki opisane we wzorze 4 dzielą układ współrzędnych na cztery ćwiartki. Zatem z rysunku 2 wynikają zależności 8, 9, 10 oraz 11.

$$x(p) \geq 0 \wedge y(p) \geq 0 \Leftrightarrow \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle \quad (8)$$

$$x(p) < 0 \wedge y(p) \geq 0 \Leftrightarrow \theta \in \left(\frac{\pi}{2}; \pi \right) \quad (9)$$

$$x(p) < 0 \wedge y(p) < 0 \Leftrightarrow \theta \in \left(\pi; \frac{3\pi}{2} \right) \quad (10)$$

$$x(p) \geq 0 \wedge y(p) < 0 \Leftrightarrow \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right\rangle \quad (11)$$

Zatem w celu udowodnienia spełnienia warunków 2 i 3 przez funkcję $\alpha(p)$ (4) należy udowodnić, że funkcja $\gamma(\theta)$ (6) jest rosnąca (jej pochodna jest większa od zera) we wszystkich przedziałach opisanych w zależnościach 8, 9, 10 oraz 11.

Obliczenia dla tych czterech przedziałów przedstawiają się w następujący sposób. Dla każdego z przedziałów możliwe jest uproszczenie funkcji $d(p)$, tak aby pozbyć się modułu, przez co różniczkowanie funkcji $\gamma(\theta)$ staje się łatwiejsze.

$$1^\circ \quad x(p) \geq 0 \wedge y(p) \geq 0 \Leftrightarrow \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$$

$$\alpha(p) = \frac{y(p)}{d(p)} = \frac{y(p)}{|x(p)| + |y(p)|} = \frac{y(p)}{x(p) + y(p)}$$

$$\gamma(\theta) = \alpha(p(\theta)) = \frac{|R| \cdot \sin \theta}{|R| \cdot \cos \theta + |R| \cdot \sin \theta} = \frac{\sin \theta}{\sin \theta + \cos \theta}$$

$$\begin{aligned} \frac{d}{d\theta} \gamma(\theta) &= \frac{d}{d\theta} \left(\frac{\sin \theta}{\sin \theta + \cos \theta} \right) \\ &= \frac{\frac{d}{d\theta}(\sin \theta) \cdot (\sin \theta + \cos \theta) - \sin \theta \cdot \frac{d}{d\theta}(\sin \theta + \cos \theta)}{(\sin \theta + \cos \theta)^2} \\ &= \frac{\cos \theta \cdot (\sin \theta + \cos \theta) - \sin \theta \cdot (\cos \theta - \sin \theta)}{\sin^2 \theta + 2 \sin \theta \cos \theta + \cos^2 \theta} \\ &= \frac{\cos \theta \sin \theta + \cos^2 \theta - \sin \theta \cos \theta + \sin^2 \theta}{1 + 2 \sin \theta \cos \theta} = \frac{1}{1 + 2 \sin \theta \cos \theta} \\ &= \frac{1}{1 + \sin 2\theta} \end{aligned}$$

$$\frac{d}{d\theta} \gamma(\theta) > 0$$

$$\frac{1}{1 + \sin 2\theta} > 0$$

$$1 + \sin 2\theta > 0$$

$$\sin 2\theta > -1$$

$$\sin 2\theta \neq -1$$

$$2\theta \neq \frac{3}{2}\pi + 2k\pi$$

$$\theta \neq \frac{3}{4}\pi + k\pi; \quad k \in \mathbb{Z}$$

$$\left. \begin{array}{l} k = 0 \Rightarrow \theta \neq \frac{3}{4}\pi > \frac{\pi}{2} \\ k = -1 \Rightarrow \theta \neq -\frac{1}{4}\pi < 0 \end{array} \right\} \Rightarrow \frac{d}{d\theta} \gamma(\theta) > 0 \text{ dla } \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$$

$$\left. \begin{array}{l} \gamma(0) = \frac{\sin 0}{\sin 0 + \cos 0} = \frac{0}{0+1} = 0 \\ \gamma\left(\frac{\pi}{2}\right) = \frac{\sin \frac{\pi}{2}}{\sin \frac{\pi}{2} + \cos \frac{\pi}{2}} = \frac{1}{1+0} = 1 \end{array} \right\} \Rightarrow \gamma(\theta) \in \langle 0; 1 \rangle \text{ dla } \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$$

$$2^\circ \quad x(p) < 0 \wedge y(p) \geq 0 \Leftrightarrow \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$$

$$\begin{aligned} \alpha(p) &= 2 - \frac{y(p)}{d(p)} = 2 - \frac{y(p)}{|x(p)| + |y(p)|} = 2 - \frac{y(p)}{-x(p) + y(p)} \\ &= 2 - \frac{y(p)}{y(p) - x(p)} \end{aligned}$$

$$\gamma(\theta) = \alpha(p(\theta)) = 2 - \frac{|R| \cdot \sin \theta}{|R| \cdot \sin \theta - |R| \cdot \cos \theta} = 2 - \frac{\sin \theta}{\sin \theta - \cos \theta}$$

$$\begin{aligned} \frac{d}{d\theta} \gamma(\theta) &= \frac{d}{d\theta} \left(2 - \frac{\sin \theta}{\sin \theta - \cos \theta} \right) \\ &= - \frac{\frac{d}{d\theta}(\sin \theta) \cdot (\sin \theta - \cos \theta) - \sin \theta \cdot \frac{d}{d\theta}(\sin \theta - \cos \theta)}{(\sin \theta - \cos \theta)^2} \\ &= - \frac{\cos \theta \cdot (\sin \theta - \cos \theta) - \sin \theta \cdot (\cos \theta + \sin \theta)}{\sin^2 \theta - 2 \sin \theta \cos \theta + \cos^2 \theta} \\ &= - \frac{\cos \theta \sin \theta - \cos^2 \theta - \sin \theta \cos \theta - \sin^2 \theta}{1 - 2 \sin \theta \cos \theta} \\ &= - \frac{-\cos^2 \theta - \sin^2 \theta}{1 - 2 \sin \theta \cos \theta} = \frac{\sin^2 \theta + \cos^2 \theta}{1 - \sin 2\theta} = \frac{1}{1 - \sin 2\theta} \end{aligned}$$

$$\frac{d}{d\theta} \gamma(\theta) > 0$$

$$\frac{1}{1 - \sin 2\theta} > 0$$

$$1 - \sin 2\theta > 0$$

$$-\sin 2\theta > -1$$

$$\sin 2\theta < 1$$

$$\sin 2\theta \neq 1$$

$$2\theta \neq \frac{\pi}{2} + 2k\pi$$

$$\theta \neq \frac{\pi}{4} + k\pi; \quad k \in \mathbb{Z}$$

$$\left. \begin{aligned} k=0 &\Rightarrow \theta \neq \frac{\pi}{4} < \frac{\pi}{2} \\ k=1 &\Rightarrow \theta \neq \frac{5}{4}\pi > \pi \end{aligned} \right\} \Rightarrow \frac{d}{d\theta} \gamma(\theta) > 0 \text{ dla } \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$$

$$\left. \begin{aligned} \gamma\left(\frac{\pi}{2}\right) &= 2 - \frac{\sin \frac{\pi}{2}}{\sin \frac{\pi}{2} - \cos \frac{\pi}{2}} = 2 - \frac{1}{1-0} = 1 \\ \gamma(\pi) &= 2 - \frac{\sin \pi}{\sin \pi - \cos \pi} = 2 - \frac{0}{0-(-1)} = 2 \end{aligned} \right\} \Rightarrow \gamma(\theta) \in (1; 2) \text{ dla } \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$$

$$3^\circ \quad x(p) < 0 \wedge y(p) < 0 \Leftrightarrow \theta \in \left(\pi; \frac{3\pi}{2}\right)$$

$$\begin{aligned} \alpha(p) &= 2 + \frac{|y(p)|}{d(p)} = 2 + \frac{-y(p)}{|x(p)| + |y(p)|} = 2 - \frac{y(p)}{-x(p) - y(p)} \\ &= 2 + \frac{y(p)}{x(p) + y(p)} \end{aligned}$$

$$\gamma(\theta) = \alpha(p(\theta)) = 2 + \frac{|R| \cdot \sin \theta}{|R| \cdot \cos \theta + |R| \cdot \sin \theta} = 2 + \frac{\sin \theta}{\sin \theta + \cos \theta}$$

$$\begin{aligned} \frac{d}{d\theta} \gamma(\theta) &= \frac{d}{d\theta} \left(2 + \frac{\sin \theta}{\sin \theta + \cos \theta} \right) \\ &= \frac{\frac{d}{d\theta}(\sin \theta) \cdot (\sin \theta + \cos \theta) - \sin \theta \cdot \frac{d}{d\theta}(\sin \theta + \cos \theta)}{(\sin \theta + \cos \theta)^2} \\ &= \frac{\cos \theta \cdot (\sin \theta + \cos \theta) - \sin \theta \cdot (\cos \theta - \sin \theta)}{\sin^2 \theta + 2 \sin \theta \cos \theta + \cos^2 \theta} \\ &= \frac{\cos \theta \sin \theta + \cos^2 \theta - \sin \theta \cos \theta + \sin^2 \theta}{1 + 2 \sin \theta \cos \theta} \\ &= \frac{\cos^2 \theta + \sin^2 \theta}{1 + 2 \sin \theta \cos \theta} = \frac{1}{1 + \sin 2\theta} \end{aligned}$$

$$\frac{d}{d\theta} \gamma(\theta) > 0$$

$$\frac{1}{1 + \sin 2\theta} > 0$$

$$1 + \sin 2\theta > 0$$

$$\sin 2\theta > -1$$

$$\sin 2\theta \neq -1$$

$$2\theta \neq \frac{3}{2}\pi + 2k\pi$$

$$\theta \neq \frac{3}{4}\pi + k\pi; \quad k \in \mathbb{Z}$$

$$\left. \begin{aligned} k=0 &\Rightarrow \theta \neq \frac{3}{4}\pi < \pi \\ k=1 &\Rightarrow \theta \neq \frac{7}{4}\pi > \frac{3\pi}{2} \end{aligned} \right\} \Rightarrow \frac{d}{d\theta} \gamma(\theta) > 0 \text{ dla } \theta \in \left(\pi; \frac{3\pi}{2}\right)$$

$$\left. \begin{aligned} \gamma(\pi) &= 2 + \frac{\sin \pi}{\sin \pi + \cos \pi} = 2 - \frac{0}{0-1} = 2 \\ \gamma\left(\frac{3\pi}{2}\right) &= 2 + \frac{\sin \frac{3\pi}{2}}{\sin \frac{3\pi}{2} + \cos \frac{3\pi}{2}} \\ &= 2 + \frac{-1}{-1+0} = 2 + 1 = 3 \end{aligned} \right\} \Rightarrow \gamma(\theta) \in (2; 3) \text{ dla } \theta \in \left(\pi; \frac{3\pi}{2}\right)$$

$$4^\circ \quad x(p) \geq 0 \wedge y(p) < 0 \Leftrightarrow \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right\rangle$$

$$\alpha(p) = 4 - \frac{|y(p)|}{d(p)} = 4 - \frac{-y(p)}{|x(p)| + |y(p)|} = 4 + \frac{y(p)}{x(p) - y(p)}$$

$$\gamma(\theta) = \alpha(p(\theta)) = 4 + \frac{|R| \cdot \sin \theta}{|R| \cdot \cos \theta - |R| \cdot \sin \theta} = 4 + \frac{\sin \theta}{\cos \theta - \sin \theta}$$

$$\begin{aligned} \frac{d}{d\theta} \gamma(\theta) &= \frac{d}{d\theta} \left(4 + \frac{\sin \theta}{\cos \theta - \sin \theta} \right) \\ &= \frac{\frac{d}{d\theta}(\sin \theta) \cdot (\cos \theta - \sin \theta) - \sin \theta \cdot \frac{d}{d\theta}(\cos \theta - \sin \theta)}{(\cos \theta - \sin \theta)^2} \\ &= \frac{\cos \theta \cdot (\cos \theta - \sin \theta) - \sin \theta \cdot (-\sin \theta - \cos \theta)}{\cos^2 \theta - 2 \cos \theta \sin \theta + \sin^2 \theta} \\ &= \frac{\cos^2 \theta - \cos \theta \sin \theta + \sin^2 \theta + \sin \theta \cos \theta}{1 - 2 \sin \theta \cos \theta} \\ &= \frac{\cos^2 \theta + \sin^2 \theta}{1 - 2 \sin \theta \cos \theta} = \frac{1}{1 - \sin 2\theta} \end{aligned}$$

$$\frac{d}{d\theta} \gamma(\theta) > 0$$

$$\frac{1}{1 - \sin 2\theta} > 0$$

$$1 - \sin 2\theta > 0$$

$$-\sin 2\theta > -1$$

$$\sin 2\theta < 1$$

$$\sin 2\theta \neq 1$$

$$2\theta \neq \frac{\pi}{2} + 2k\pi$$

$$\theta \neq \frac{\pi}{4} + k\pi; \quad k \in \mathbb{Z}$$

$$\left. \begin{aligned} k=1 &\Rightarrow \theta \neq \frac{5\pi}{4} > \frac{3\pi}{2} \\ k=2 &\Rightarrow \theta \neq \frac{9\pi}{4} > 2\pi \end{aligned} \right\} \Rightarrow \frac{d}{d\theta} \gamma(\theta) > 0 \text{ dla } \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right\rangle$$

$$\left. \begin{aligned} \gamma\left(\frac{3\pi}{2}\right) &= 4 + \frac{\sin \frac{3\pi}{2}}{\cos \frac{3\pi}{2} - \sin \frac{3\pi}{2}} \\ &= 4 + \frac{-1}{0 - (-1)} = 4 - 1 = 3 \\ \gamma(2\pi) &= 4 + \frac{\sin 2\pi}{\cos 2\pi - \sin 2\pi} \\ &= 4 + \frac{0}{1 - 0} = 4 \end{aligned} \right\} \Rightarrow \gamma(\theta) \in \langle 3; 4 \rangle \text{ dla } \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right\rangle$$

Z powyższych obliczeń wynika, że we wszystkich przedziałach z zależności 8, 9, 10 oraz 11 funkcja $\gamma(\theta)$ ma dodatnią pochodną, a tym samym jest rosnąca we wszystkich przedziałach, oraz każda z wartości osiąganych w przedziale jest większa od każdej wartości z poprzedniego przedziału, a tym samym funkcja $\alpha(p)$ spełnia warunki 2 i 3.

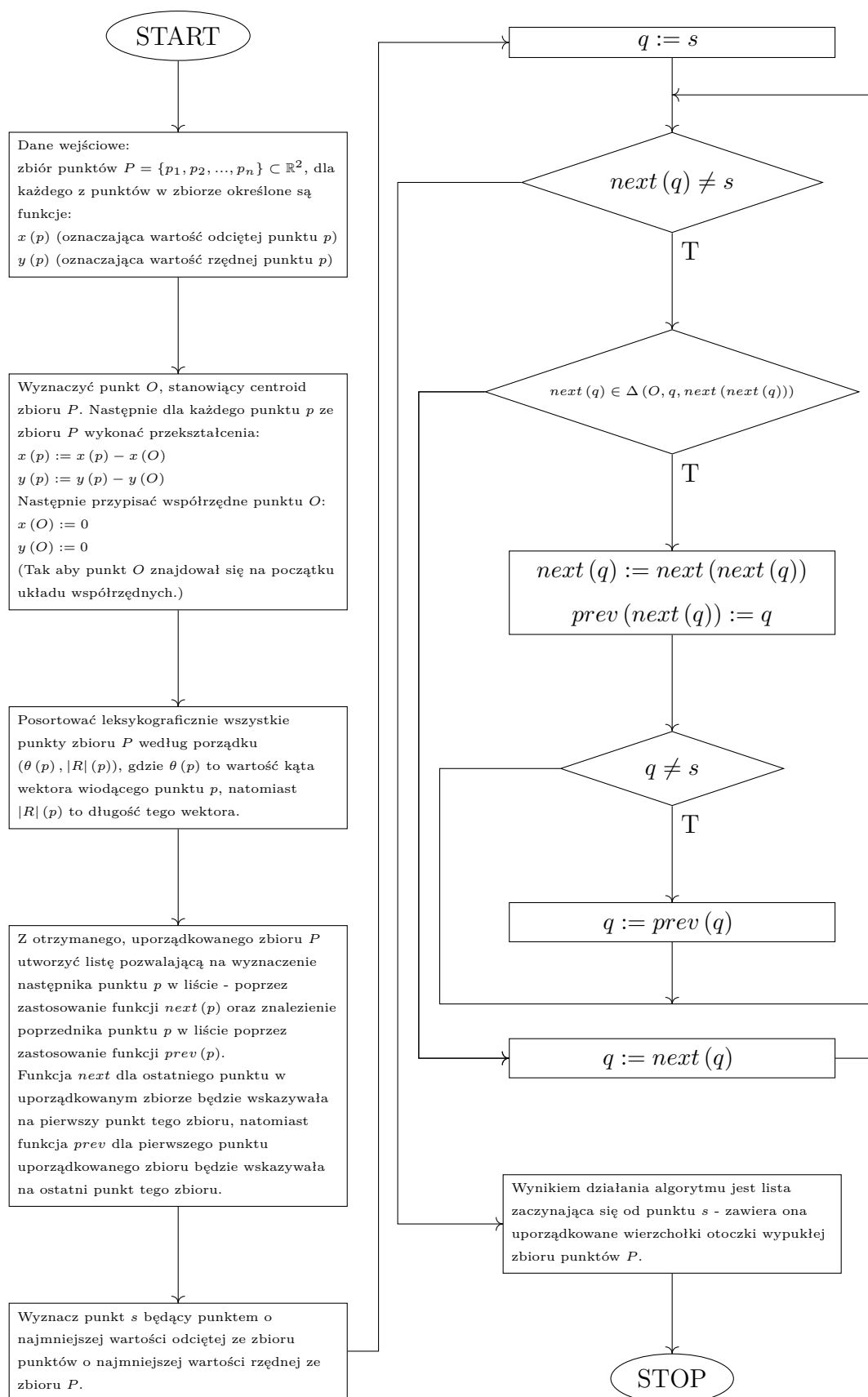
Postać funkcji $\gamma(\theta)$ może być opisana wzorem 12 a wykres jej przebiegu, przedstawiony na rysunku 3 wyraźnie potwierdza jej rosnącą monotoniczność we wszystkich przedziałach.

$$\gamma(\theta) = \begin{cases} \frac{\sin \theta}{\sin \theta + \cos \theta} & \text{dla } \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle \\ 2 - \frac{\sin \theta}{\sin \theta - \cos \theta} & \text{dla } \theta \in \left(\frac{\pi}{2}; \pi \right) \\ 2 + \frac{\sin \theta}{\sin \theta + \cos \theta} & \text{dla } \theta \in \left(\pi; \frac{3\pi}{2} \right) \\ 4 - \frac{\sin \theta}{\sin \theta + \cos \theta} & \text{dla } \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right\rangle \end{cases} \quad (12)$$



Rysunek 3: Wykres funkcji $\gamma(\theta)$
Źródło: opracowanie własne

1.1 Algorytm Grahama



Rysunek 4: Algorytm Grahama

1.2 Algorytm Jarvisa

2. Otoczka wypukła wielokąta prostego

[1]

3. Redukcja zbioru punktów do wielokąta prostego

Rozdział II

Zastosowania

Istotnym, jeśli nie najistotniejszym zagadnieniem dotyczącym otoczek wypukłych są ich zastosowania. Już od kilkudziesięciu lat problem ten temat znajduje swoje użycie w wielu dziedzinach kombinatoryki i informatyki. Dla przykładu problem sortowania elementów liczbowych listy można sprowadzić do problemu znalezienia otoczki wypukłej. W związku z tym rozwiązanie tego problemu może pomóc w rozwiązaniu innych problemów w bardziej symboliczny i graficzny sposób.

1. Generalizacja kartograficzna

W celu jak najbardziej informatywnego i niezłożonego przedstawienia danych geograficznych w sposób graficzny potrzebne jest zastosowanie algorytmu generalizującego informacje. Taka generalizacja w kartografii zaczęła być coraz bardziej potrzebna, kiedy w dobie cyfrowego przetwarzania danych mapy zaczęły być przechowywane cyfrowo. Przed tym generalizacją zajmowali się manualnie ludzie, którzy wyposażeni w ludzkie zmysły wykrywania wzorów i kształtów byli w stanie z łatwością narysować obrys zbioru drzew na płaszczyźnie i opisać go jako teren leśny. W przypadku przechowywania danych geograficznych w postaci cyfrowej zaistniała potrzeba wykonywania tej samej operacji w sposób zalgorytmizowany. [2]

Do celu generalizacji można by wykorzystać otoczkę wypukłą w sposób bezpośredni [3], jednak taki obrys terenu może zawierać w sobie przestrzenie pozbawione obiektów, co byłoby niepomocne dla ludzkiej interpretacji przedstawianego mu obszaru.

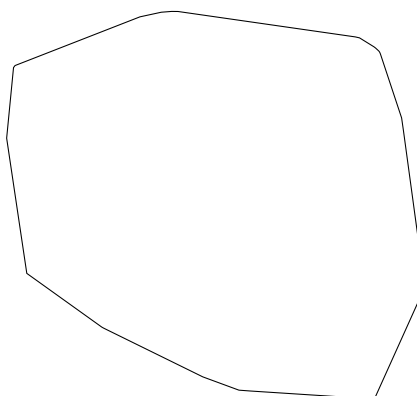
Należy też zwrócić uwagę, że otoczka wypukła wielokąta prostego zawiera w sobie wszystkie punkty należące do wewnętrznej powierzchni tego wielokąta. [1] W związku z tym jeśli udowodni się, że dany punkt nie należy do wnętrza otoczki wypukłej będzie to implikowało, że punkt ten nie należy do wnętrza wielokąta prostego. Taka wiedza okazuje się być przydatna jeśli uwzględnimy dodatkowo fakt, że określanie czy punkt leży wewnątrz wielokąta wypukłego jest dużo łatwiejsze niż dla pozostałych wielokątów. [4]

Dla przykładu rozpatrując obrys granic Polski (Rysunek 5; źródło współrzędnych obrysu: [5]) można zauważyć, że jego kształt jest bardzo skomplikowany. W istocie zawiera on 1336 krawędzi.



Rysunek 5: Obrys granic Polski
Źródło: opracowanie własne

Natomiast otoczka wypukła obrysu granic Polski zawiera tylko 28 krawędzi i została przedstawiona na rysunku 6.



Rysunek 6: Otoczka wypukła obrysu granic Polski
Źródło: opracowanie własne

W związku z tym, że złożoność obliczeniowa algorytmu sprawdzającego czy punkt należy do wnętrza danego wielokąta jest liniowa [6], możliwe jest dużo szybsze wykluczenie punktów znajdujących się poza otoczką.

2. Grafika komputerowa

Obiekty wykorzystywane w grafice komputerowej często mogą charakteryzować się skomplikowanymi kształtami. Im bardziej skomplikowany kształt, tym więcej mocy

obliczeniowej potrzeba w celu wykonania danej operacji na tym kształcie. W niektórych przypadkach do uproszczenia graficznego danego obiektu używa się otoczki wypukłej jego kształtu. Ze względu na fakt, iż otoczka wypukła wielokąta zawsze będzie miała liczbę wierzchołków mniejszą lub równą liczbie wierzchołków samego wielokąta, powstała otoczka może posłużyć do wykonania mniejszej liczby obliczeń przy wykonywaniu operacji na danym obiekcie.

3. Detekcja obiektów

Proces analizowania obrazu przez ludzkie organy jest wyjątkowo złożony złożony [7]. W celu imitacji tego procesu w sposób algorytmiczny, zostały przeprowadzone liczne badania naukowe [8]. Dzięki temu zaproponowano wiele podejść rozwiązania tego problemu. Istotnym aspektem ze względów praktycznych tych pomysłów jest złożoność obliczeniowa ich implementacji.

Jednym z działań wykonywanych przez ludzkie zdolności poznawcze po zarejestrowaniu obrazu jest wybranie miejsca w obrazie o największym znaczeniu (z ludzkiego punktu widzenia) [9]. Niektóre z rozwiązań tego problemu w informatyce opierały się nie tylko na samych danych pochodzących z obrazu, ale również tworząc modele z uwzględnieniem danych statycznych z badań dotyczących ludzkich umiejętności poznawczych [10]. Jednak spora część z nich opiera się na przetwarzaniu samych danych pozyskanych z obrazów, bez potrzeby dostarczania danych statycznych. Aby komputer był w stanie wykonać tego typu operację potrzebny jest szereg złożonych obliczeniowo operacji [8], [10].

W celu zmniejszenia kosztu obliczeniowego tego typu procesu można wykonać preprocesowanie danych wejściowych. Jednym z nich może być wykorzystanie algorytmu wyznaczającego otoczkę wypukłą tak zwanych punktów kluczowych [8].

4. Wyznaczanie obwiedni sygnału

Sygnał to zapis informacji zmieniającej się w zależności od czasu. Taką informacją może być natężenie/napięcie prądu, natężenie pola elektromagnetycznego, dźwięk utworu muzycznego itd. W przypadku, jeśli zmienną informację da się reprezentować w sposób liczbowy, możliwe jest graficzne przedstawienie sygnału. Zapisanie informacji w ten sposób może być łatwiejsze do interpretacji przez człowieka. Jednak

w przypadku, gdy mamy do czynienia ze skomplikowanym przebiegiem, na

Algorytm wyznaczający otoczkę wypukłą wielokąta prostego może posłużyć
znajdowaniu obwiedni sygnału.

Rozdział III

Implementacja w języku Scala

1. Pojęcia ogólne

W tej sekcji przedstawione zostaną listingi zawierające kod w języku Scala definiujący pojęcia potrzebne do implementacji poszczególnych algorytmów wyznaczających otoczkę wypukłą.

Klasa `Point` (Listing 1: linia 4) posłuży jako podstawowa klasa reprezentująca punkt w przestrzeni dwuwymiarowej. W celu łatwiejszego manipulowania danymi. Zostały w jej ramach zaimplementowane 3 metody pomocnicze.

Odejmowanie (Listing 1: linia 7) - zaimplementowane jako odejmowanie od siebie odpowiadających sobie współrzędnych dwu punktów.

Dodawanie (Listing 1: linia 8) - dodawanie do siebie odpowiadających sobie współrzędnych dwu punktów.

Dzielenie przez skalar (Listing 1: linia 9) - zaimplementowane jako dzielenie obu współrzędnych przez wskazaną liczbę. Zaimplementowane dzięki użyciu domniemanemu argumentowi `numeric: Numeric[T]`. W języku Scala tego typu konstrukcja umożliwia korzystanie z metody `/` w tedy i tylko wtedy jeśli zaimportowany zostanie do aktualnego kontekstu obiekt typu `Numeric[T]`. Domyślnie typ ten jest zaimplementowany dla `T` będącym jednym z podstawowych typów liczbowych: `Float`, `Int`, `Double` etc. Więc w domyśle tego typu konstrukcja służy niejako „udowodnieniu”, że wskazywany argument jest w istocie liczbą.

```
1 package utilities.geometry
2
3 case class Point(
4     x: Double,
5     y: Double
6 ) {
7     def -(that: Point): Point = Point(x - that.x, y - that.y)
8     def +(that: Point): Point = Point(x + that.x, y + that.y)
9     def /[T](number: T)(implicit numeric: Numeric[T]): Point = {
10         val double = numeric.toDouble(number)
```

```

11     Point(x / double, y / double)
12 }
13 }

```

Listing 1: Point.scala

Źródło: opracowanie własne

Kolejnym istotnym elementem wykorzystanym w implementacji algorytmów wyznaczających otoczkę wypukłą zbioru punktów na płaszczyźnie są metody wyliczające określone właściwości punktów.

Metoda `calculateCentroid(points: Points): Points` (Listing 2: linia 11) wyznacza centroid zbioru punktów `points`.

Metoda `distanceFromCenterSquared(p: Point): Double` (Listing 2: linia 17) wyznacza kwadrat odległości wskazanego punktu od środka układu współrzędnych.

Na podstawie wyżej wymienionej metody zostały zaimplementowane trzy metody związane z odległością między punktami.

Metoda `distanceSquared(a: Point, b: Point): Double` (Listing 2: linia 13) wyznacza kwadrat odległości pomiędzy dwoma punktami.

Metoda `distanceFromCenter(p: Point): Double` (Listing 2: linia 19) wyznacza odległość od środka układu współrzędnych.

Metoda `distance(a: Point, b: Point): Double` (Listing 2: linia 15) wyznacza odległość pomiędzy dwoma punktami.

W ramach obiektu `PointsUtils` (Listing 2: linia 5) zostały również zaimplementowane metody służące do wyznaczania wartości umożliwiających sortowanie punktów względem kąta występującego pomiędzy horyzontalną osią układu współrzędnych, a wektorem łączącym początek układu współrzędnych z punktem (kąt θ zaznaczony na rysunku 2).

Metoda `phase(p: Point): Double` (Listing 2: linia 21) wyznacza dokładną wartość szukanego kąta. Matematycznie ta funkcja została zapisana we wzorze 1.

Metoda `alpha(p: Point): Double` (Listing 2: linia 32) wyznacza wartość umożliwiającą porównanie kąta dla wskazanego punktu z innymi punktami. Matematycznie ta funkcja została zapisana we wzorze 4.

```

1 package utilities.geometry

```

```

2

```

```

3 import java.lang.Math._
4
5 object PointsUtils {
6
7     type Points = List[Point]
8
9     private lazy val HALF_PI = PI * .5
10
11     def calculateCentroid(points: Points): Point = points.reduceLeft(
12         _ + _) / points.length
13
14     def distanceSquared(a: Point, b: Point): Double =
15         distanceFromCenterSquared(a - b)
16
17     def distance(a: Point, b: Point): Double = distanceFromCenter(a -
18         b)
19
20     def distanceFromCenterSquared(p: Point): Double = pow(p.x, 2) +
21         pow(p.y, 2)
22
23     def distanceFromCenter(p: Point): Double = sqrt(
24         distanceFromCenterSquared(p))
25
26     def phase(p: Point): Double = {
27         import p.{x, y}
28         (x.sign, y.sign) match {
29             case (1, _) => atan(y / x)
30             case (-1, 0 | 1) => atan(y / x) + PI
31             case (-1, -1) => atan(y / x) - PI
32             case (0, -1) => HALF_PI
33             case (0, 1) => -HALF_PI
34         }
35     }
36
37     def alpha(p: Point): Double = {
38         import p.{x, y}
39         val absX = abs(x)
40         val absY = abs(y)
41         val d = absX + absY

```

```
37
38 (x.sign, y.sign) match {
39     case (0 | 1, 0 | 1) => y / d
40     case (-1, 0 | 1) => 2 - y / d
41     case (-1, -1) => 2 + absY / d
42     case (0 | 1, -1) => 4 - absY / d
43 }
44 }
45 }
```

Listing 2: PointsUtils.scala

W celu zapewnienia odpowiedniego sortowania punktów został utworzony `trait OrientationOrdering`. Język Scala umożliwia wykorzystywanie zaimplementowanie własnego obiektu `Ordering[T]`, który będzie następnie używany w metodzie `sorted` dla kolekcji `C[T]`.

Trait `OrientationOrdering` (Listing 3: linia 6) ma zaimplementowaną metodę `compare(x: Point, y: Point)` (Listing 3: linia 7) w taki sposób, aby w pierwszej kolejności porównywane były wartości funkcji reprezentujących fazę tych punktów, a następnie porównywane wartości funkcji reprezentujących odległość od środka układu współrzędnych.

W ramach `OrientationOrdering` zostały zaimplementowane dwa obiekty.

Obiekt `Exact` (Listing 3: linia 19) służy do dokładnego porównania fazy punktów, wyznaczonej ze wzoru 1 oraz odległości od środka układu współrzędnych.

Obiekt `Indicator` (Listing 3: linia 24) służy do porównywania fazy punktów za pomocą funkcji wyznaczonej ze wzoru 4.

```

1 package utilities.geometry.ordering
2
3 import utilities.geometry.{Point, PointsUtils}
4 import Ordering.Double.TotalOrdering
5
6 trait OrientationOrdering extends Ordering[Point] {
7     final override def compare(x: Point, y: Point): Int = {
8         TotalOrdering.compare(phase(x), phase(y)) match {
9             case 0 => TotalOrdering.compare(distance(x), distance(y))
10            case phaseCompared => phaseCompared
11        }
12    }
13
14    protected def distance(p: Point): Double
15    protected def phase(p: Point): Double
16 }
17
18 object OrientationOrdering {
19     implicit object Exact extends OrientationOrdering {
20         override protected def distance(p: Point): Double = PointsUtils
            .distanceFromCenter(p)
21         override protected def phase(p: Point): Double = PointsUtils.

```

```
    phase(p)
22 }
23
24 implicit object Indicator extends OrientationOrdering {
25     override protected def distance(p: Point): Double = PointsUtils
26     .distanceFromCenterSquared(p)
27     override protected def phase(p: Point): Double = PointsUtils.
28     alpha(p)
29 }
```

Listing 3: OrientationOrdering.scala

```
1 package utilities.geometry.convexhull.algorithms
2
3 import utilities.geometry.PointsUtils.Points
4 import utilities.geometry.ordering.OrientationOrdering
5
6 trait ConvexHullAlgorithm extends Product {
7   def calculate(points: Points)(implicit ordering:
8     OrientationOrdering): Points
9 }
```

Listing 4: ConvexHullAlgorithm.scala


```

1 package utilities.geometry.convex.hull.algorithms
2
3 import org.scalatest.Assertion
4 import org.scalatest.wordspec.AnyWordSpec
5 import utilities.geometry.Point
6 import utilities.geometry.PointsUtils.Points
7 import AlgorithmTest.{EXPECTED_CONVEX_HULL, INPUT_POINTS,
8   ONE_INPUT_POINT}
9 import utilities.geometry.convexhull.algorithms.ConvexHullAlgorithm
10 import utilities.geometry.ordering.OrientationOrdering
11
12 abstract class AlgorithmTest(convexHullAlgorithm:
13   ConvexHullAlgorithm)(implicit ordering: OrientationOrdering)
14   extends AnyWordSpec {
15     private def convexHullAlgorithmName: String = convexHullAlgorithm
16       .productPrefix
17
18     private def assertExpectedOutput(input: Points, expectedOutput:
19       Points): Assertion = {
20       val actualConvexHull = convexHullAlgorithm.calculate(input)
21       assert(actualConvexHull == expectedOutput)
22     }
23
24     s"$convexHullAlgorithmName" must {
25       "throw the IllegalArgumentException for empty" in {
26         assertThrows[IllegalArgumentException](convexHullAlgorithm.
27           calculate(Nil))
28       }
29
30       "return the same thing for one input point" in {
31         assertExpectedOutput(ONE_INPUT_POINT, ONE_INPUT_POINT)
32       }
33
34       "properly calculate convex hull" in {
35         assertExpectedOutput(INPUT_POINTS, EXPECTED_CONVEX_HULL)
36       }
37     }
38 }

```

```
34
35 object AlgorithmTest {
36   val INPUT_POINTS: Points = List(
37     Point(1, 3),
38     Point(2, 4),
39     Point(2, 2),
40     Point(4, 6),
41     Point(3, 4),
42     Point(4, 4),
43     Point(4, 3),
44     Point(2, 6),
45     Point(5, 2),
46     Point(6, 4),
47     Point(5, 5)
48   )
49
50   val EXPECTED_CONVEX_HULL: Points = List(
51     Point(6, 4),
52     Point(4, 6),
53     Point(2, 6),
54     Point(1, 3),
55     Point(2, 2),
56     Point(5, 2)
57   )
58
59   val ONE_INPUT_POINT: Points = List(
60     Point(1, 2)
61   )
62 }
```

Listing 5: AlgorithmTest.scala

2. Algorytm Grahama

```
1 package utilities.geometry.convexhull.algorithms
2 import utilities.geometry.PointsUtils.Points
3 import utilities.geometry.ordering.OrientationOrdering
4 import utilities.geometry.{PointsCycle, PointsUtils}
5
6
7 case object Graham extends ConvexHullAlgorithm {
8
9   override def calculate(points: Points)(implicit ordering:
10     OrientationOrdering): Points = {
11     require(points.nonEmpty, "You can not calculate convex hull of
12       an empty points set.")
13     val centroid = PointsUtils.calculateCentroid(points)
14     val pointsSorted = points.map(_ - centroid).sorted
15     val pointsCycle = new PointsCycle(pointsSorted)
16
17     pointsCycle.getHull.map(_ + centroid)
18   }
19 }
```

Listing 6: Graham.scala

```

1 package utilities.geometry
2
3 import utilities.geometry.PointsUtils.Points
4
5 class PointsCycle(points: Points) extends Iterable[Point] {
6
7     private lazy val first = NoPointRef.next
8
9     override def iterator: Iterator[Point] = new Iterator[Point] {
10         private var currentPointRef: AbstractPointRef = NoPointRef
11
12         override def hasNext: Boolean = currentPointRef.isNotLast
13
14         override def next(): Point = {
15             val next = currentPointRef.next
16             currentPointRef = next
17             next.point
18         }
19     }
20
21     private def createPointRefs(points: Points): AbstractPointRef =
22         points.foldLeft[AbstractPointRef](NoPointRef)(_ . addPoint(_)) .
23             addLastPointAndReturn()
24
25     createPointRefs(points)
26
27     trait AbstractPointRef {
28         var next: PointRef = _
29         var prev: PointRef = _
30
31         def isNotFirst: Boolean
32         def isNotLast: Boolean
33
34         def addPoint(point: Point): PointRef
35
36         protected def addLastPoint(): Unit
37
38         final def addLastPointAndReturn(): AbstractPointRef = {
39             addLastPoint()

```

```

39     this
40   }
41 }
42
43 class PointRef(val point: Point) extends AbstractPointRef {
44
45   final def isNotFirst: Boolean = this ne first
46   final def isNotLast: Boolean = next ne first
47
48   override def addPoint(point: Point): PointRef = {
49     val pointRef = new PointRef(point)
50     pointRef.prev = this
51     next = pointRef
52     pointRef
53   }
54
55   def isInTriangle(t2: PointRef, t3: PointRef): Boolean =
56     Orientation.calculateThreePointsOrientation(t2.point, t3.
57 point, point) == Orientation.Left
58
59   override protected def addLastPoint(): Unit = {
60     next = first
61     first.prev = this
62   }
63 }
64
65 object NoPointRef extends AbstractPointRef {
66
67   override def addPoint(point: Point): PointRef = {
68     val pointRef = new PointRef(point)
69     next = pointRef
70     pointRef
71   }
72
73   override def isNotFirst: Boolean = first ne null
74   override def isNotLast: Boolean = first ne null
75
76   override protected def addLastPoint(): Unit = ()
77 }

```

```
77
78 def getHull: Points = {
79
80   if (NoPointRef.isNotLast) {
81     var q = NoPointRef.next
82     while (q.isNotLast) {
83       if (q.next.isInTriangle(q, q.next.next)) {
84         q.next = q.next.next
85         q.next.prev = q
86         if (q.isNotFirst) q = q.prev
87       } else q = q.next
88     }
89   }
90   toList
91 }
92
93 }
```

Listing 7: PointsCycle.scala

```

1 package utilities.geometry.convex.hull.algorithms
2
3 import utilities.geometry.convexhull.algorithms.Graham
4 import utilities.geometry.ordering.OrientationOrdering.Indicator
5
6 class GrahamAlgorithmTest extends AlgorithmTest(Graham)

```

Listing 8: GrahamAlgorithmTest.scala

3. Algorytm Jarvisa

Dzięki wykorzystaniu pattern matchingu dostępnego w Scali, możliwe jest przejrzyste zaimplementowanie algorytmu Jarvisa, który, jak widać jest dużo mniej skomplikowany od algorytmu Grahama. Mniejsze skomplikowanie implementacji wiąże się jednak z dużo większą złożonością obliczeniową.

```

1 package utilities.geometry.convexhull.algorithms
2
3 import utilities.geometry.{Point, PointsUtils}
4 import utilities.geometry.PointsUtils.Points
5
6 case class Jarvis() extends ConvexHullAlgorithm {
7
8     private val FIRST_POINT_ORDERING = new Ordering[Point] {
9         override def compare(x: Point, y: Point): Int = {
10             val byX = Ordering.Double.TotalOrdering.compare(x.x, y.x)
11             lazy val byY = Ordering.Double.TotalOrdering.compare(x.y, y.y)
12             if (byX == 0) byY
13             else byX
14         }
15     }
16
17     override protected def nonEmptyCalculate(points: Points): Points
18     = {
19         val firstPoint = points.max(FIRST_POINT_ORDERING)
20         val pZero = Point(firstPoint.x - 1, firstPoint.y)
21
22         var pI = firstPoint

```

```

23     var pIMinusOne = pZero
24
25     val pointsBuilder = List.newBuilder[Point]
26     pointsBuilder += firstPoint
27
28     while (true) {
29         points
30             .filterNot(_ == pI)
31             .maxByOption(PointsUtils.angleBetweenThreePoints(pIMinusOne
, pI, _)) match {
32             case Some('firstPoint') => return pointsBuilder.result()
33             case Some(pIPlusOne) =>
34                 pointsBuilder += pIPlusOne
35                 pIMinusOne = pI
36                 pI = pIPlusOne
37             case _ => return pointsBuilder.result()
38         }
39     }
40     Nil
41 }
42
43 }

```

Listing 9: Jarvis.scala

```

1 package utilities.geometry.convex.hull.algorithms
2
3 import utilities.geometry.convexhull.algorithms.Jarvis
4
5 class JarvisAlgorithmTest extends AlgorithmTest(Jarvis())

```

Listing 10: JarvisAlgorithmTest.scala

Rozdział IV

Dynamiczna otoczka wypukła

1. Algorytm
2. Implementacja w języku Scala

Rozdział V

Podsumowanie

Bibliografia

- [1] R. L. Graham i F. Yao, *Finding the Convex Hull of a Simple Polygon*. 1981.
- [2] J. Jourban i Y. Gabay, *A Method for Construction of 2D Hull For Generalized Cartographic Representation*. 2000.
- [3] R. Laurini, *Geographic Knowledge Infrastructure*. 2017, s. 139–156.
- [4] P. Heckbert, *Graphics Gems IV* (Graphics gems series). 1994, s. 30.
- [5] Datopian, *Country Polygons as GeoJSON* — *datahub.io*,
<https://datahub.io/core/geo-countries#resource-countries>,
 [Data dostępu: 13:34 20.08.2023].
- [6] C.-W. Huang i T.-Y. Shih, *On The Complexity Of Point-In-Polygon Algorithms*. 1996, s. 109–118.
- [7] I. Biederman, *Human image understanding: Recent research and a theory*. 1985, s. 30.
- [8] N. Singh, R. Arya i R. Agrawal, *A convex hull approach in conjunction with Gaussian mixture model for salient object detection*. 2016.
- [9] D. Ndayikengurukiye i M. Mignotte, *CoSOV1Net: A Cone- and Spatial-Opponent Primary Visual Cortex-Inspired Neural Network for Lightweight Salient Object Detection*. 2023.
- [10] A. Borji, M.-M. Cheng, Q. Hou, H. Jiang i J. Li, *Salient object detection: A survey - computational visual media*. 2019.