Proceduralne generowanie treści

Zestaw 2

1. Zaimplementować algorytm generowania fraktala inwersji zbioru gwieździstego (Algorytm 1). W programie mamy mieć możliwość generowania fraktala, który dany jest przez okręgi oraz wielokąty gwieździste. Jako iterację wykorzystać iterację Manna.

Algorytm 1: Metoda generowania fraktala inwersji zbioru gwieździstego.

Dane: $S_1, \ldots, S_k$ – zbiory gwieździste z danymi środkami inwersji; $c_1, \ldots, c_k$ – kolory transformacji; p_0 – punkt startowy leżący na zewnątrz $S_1, \ldots, S_k$; $n > 20$ – liczba iteracji; P_v – iteracja z parameterami v; W, H – rozdzielczość obrazu; $\gamma \in \mathbb{R}_+$

Wynik: Obraz I z aproksymacją fraktala inwersji zbioru gwieździstego.

for $(x, y) \in \{0, 1, \ldots, W-1\} \times \{0, 1, \ldots, H-1\}$ **do**
    $I(x, y) =$ kolor czarny
    $\mathcal{H}(x, y) = 0$
$c =$ losowy kolor
$j =$ liczba losowa z $\{1, \ldots, k\}$
$p = P_v(I_{S_j}, p_0)$
for $i = 2$ **to** n **do**
    $l =$ liczba losowa z $\{1, \ldots, k\}$
    while $j = l$ **or** $inSet(S_l, p)$ **do**
        $l =$ liczba losowa z $\{1, \ldots, k\}$
    $j = l$
    $p = P_v(I_{S_j}, p)$
    if $i > 20$ **then**
        $x = \lfloor x_p \rfloor$
        $y = \lfloor y_p \rfloor$
        $\mathcal{H}(x, y) = \mathcal{H}(x, y) + 1$
        $c = \frac{c + c_j}{2}$
        $I(x, y) = c$
$m_{\mathcal{H}} = \max_{(x,y)} \mathcal{H}(x, y)$
for $(x, y) \in \{0, 1, \ldots, W-1\} \times \{0, 1, \ldots, H-1\}$ **do**
    if $\mathcal{H}(x, y) > 0$ **then**
        $I(x, y) = \left(\frac{\log_2(1 + \mathcal{H}(x,y))}{\log_2(1 + m_{\mathcal{H}})} \right)^{1/\gamma} I(x, y)$

Przy wyznaczaniu inwersji zbioru gwieździstego potrzebować będziemy znaleźć punkt przecięcia promienia z brzegiem zbioru gwieździstego. Załóżmy, że promień dany jest wzorem $r(t) = o + t(p - o)$, gdzie $r \in [0, \infty)$, o – środek inwersji, p – punkt definiujący promień.

W przypadku okręgu o środku c i promieniu r wstawiając wzór na promień do równania okręgu otrzymujemy równanie:

$$\|p - o\|^2 t^2 + 2[(p - o) \cdot (o - c)]t + \|o - c\|^2 - r^2 = 0,$$

gdzie $\cdot$ to iloczyn skalarny. Jest to równanie kwadratowe zmiennej t. Rozwiązując to równanie otrzymujemy: brak rozwiązań (brak przecięcia), jedno rozwiązanie (promień jest styczny do okręgu) lub dwa rozwiązania (ponieważ rozważamy promień, więc rozwiązaniem, które nas interesuje jest rozwiązanie dodatnie). Po znalezieniu rozwiązania t_* wstawiamy je do równania promienia $r(t_*)$ otrzymując punkt przecięcia.

W przypadku wielokąta gwieździstego, aby znaleźć punkt przecięcia promienia z brzegiem tego wielokąta po prostu sprawdzamy przecięcie promienia z każdą z krawędzi wielokąta. Ponieważ wielokąt jest gwieździsty istnieje tylko jeden taki punkt, więc w momencie znalezienia punktu przecięcia nie musimy sprawdzać kolejnych krawędzi. Do sprawdzenia przecięcia promienia z krawędzią możemy wykorzystać Algorytm 2.

Algorytm 2: Algorytm znajdowania przecięcia promień–odcinek.

Dane: p_0, p_1 – końce odcinka; o – początek promienia; p – punkt wyznaczający kiernek promienia.

Wynik: Punkt przecięcia promień–odcinek lub *null* jeśli brak przecięcia.

$d_r = p - o$
$d_s = p_1 - p_0$
$q = p_0 - o$
$d = d_{ry} d_{sx} - d_{rx} d_{sy}$
if $|d| < 0.0001$ **then**
    return *null*
$t = \frac{1}{d}(d_{sx} a_y - d_{sy} a_x)$
$s = \frac{1}{d}(d_{rx} a_y - d_{ry} a_x)$
if $t \geq 0 \wedge s \in [0, 1]$ **then**
    return $o + t d_r$
return *null*

Do pełnej implementacji Algorytmu 1 potrzebujemy jeszcze testu sprawdzającego czy dany punkt p należy do zbioru gwieździstego. W przypadku okręgu test możemy przeprowadzić stosując szkolną matematykę czyli odpowiedni warunek wynikający z równania okręgu. Dla wielokąta gwieździstego sytuacja jest nieco bardziej skomplikowana. W tym celu możemy skorzystać z Algorytmu 3.

Przykładowe zbiory gwieździste definiujące fraktale inwersji zbiorów gwieździstych znajdują się w archiwum *inv_fractals.zip*. Plik ze zbiorami ma następującą strukturę. W pierwszej linii znajduje się pojedyncza liczba k oznaczająca liczbę zbiorów gwieździstych. W kolejnych k liniach mamy definicje tych zbiorów. Każda linia zaczyna się pojedynczą literą: C – okrąg, P – wielokąt. W przypadku okręgu kolejne dwie liczby to współrzędne środka inwersji, a następne trzy to współrzędne środka okręgu i jego promień.

Algorytm 3: Test czy podany punkt leży wewnątrz wielokąta gwieździstego.

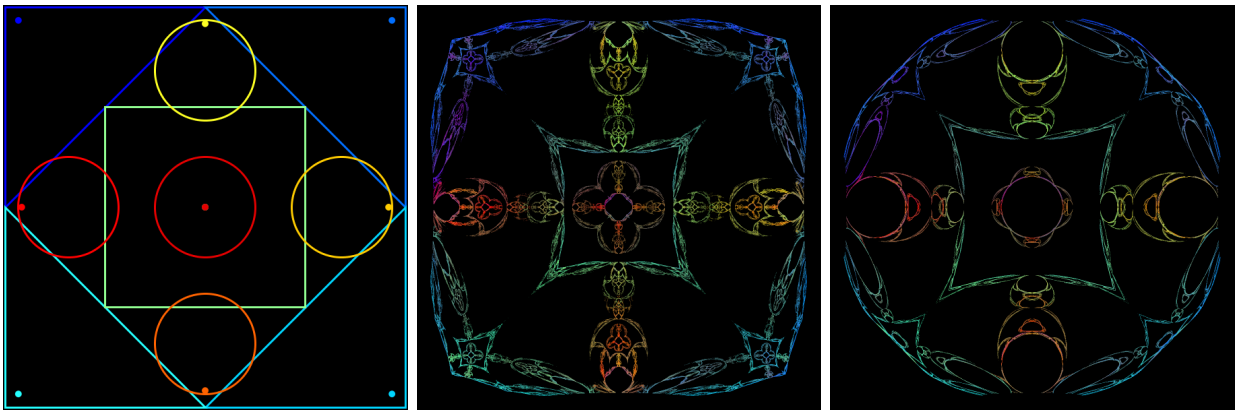
Dane: $v_0, v_1, \ldots, v_n$ – wierzchołki wielokąta; *orient* – orientacja wierzchołków (*true* – przeciwnie do ruchu wskazówek zegara, $false$ – zgodnie z ruchem wskazówek zegara); p – punkt, który testujemy.

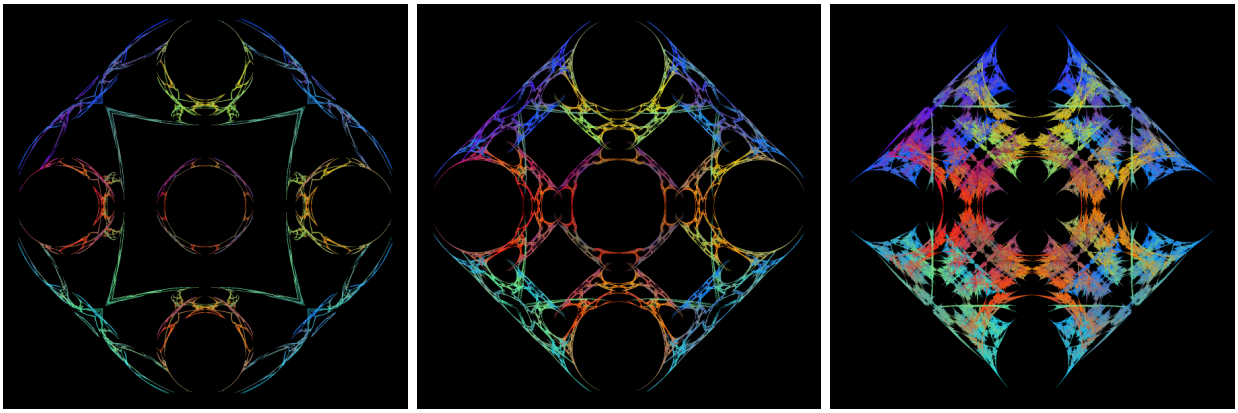
Wynik: *true* – punkt leży wewnątrz wielokąta; $flase$ – punkt leży na zewnątrz wielokąta.

```
1  inside = false
2  j = n
3  for i = 0, 1, ..., n do
4      i_0 = (orient) ? j : i
5      i_1 = (orient) ? i : j
6      if (v_{i_0 y} ≤ p_y ∧ p_y < v_{i_1 y}) ∨ (v_{i_1 y} ≤ p_y ∧ p_y < v_{i_0 y}) then
7          x = v_{i_0 x} + (p_y − v_{i_0 y})(v_{i_1 x} − v_{i_0 x}) / (v_{i_1 y} − v_{i_0 y})
8          if x > p_x then
9              inside = !inside
10     j = i
11 return inside
```

W przypadku wielokąta dwie pierwsze liczby to środek inwersji. Kolejna liczba to liczba wierzchołków wielokąta, po czym każda para liczb to współrzędne jednego wierzchołka wielokąta.

Przykład fraktala inwersji zbioru gwieździstego (*various_03.sif*) dla różnych wartości parametru α w iteracji Manna. Pierwszy obraz to zbiory gwieździste, kolejne obrazy to fraktale dla α: 1.0, 0.9, 0.8, 0.7, 0.6.





2. Zaimplementować algorytm wielomianografii z iteracjami (Algorytm 4). W programie mamy mieć możliwość wyboru metody znajdowania pierwiastków (co najmniej dwie metody, obie różne od metody Newtona), dowolnego wielomianu, rodzaju iteracji (co najmniej dwie iteracje, obie różne od iteracji Picarda) i testu zbieżności (co najmniej dwa różne testy zbieżności). Wartość wielomianu dla danego punktu należy obliczyć za pomocą schematu Hornera.

   Przykładowe metody znajdowania pierwiastków, które można wykorzystać, można znaleźć np. tutaj

   - Sekcja 5 „Newton Fraktale" http://www.3d-meier.de/tut20/Seite1.html
   - Gościniak, I., Gdawiec, K.: PSO-based Newton-like Method and Iteration Processes in the Generation of Artistic Patterns. Lecture Notes in Computer Science, vol. 11241, pp. 47-56, (2018) wersja PDF
   - Gdawiec, K., Kotarski, W., Lisowska, A.: On the Robust Newton's Method with the Mann Iteration and the Artistic Patterns from Its Dynamics. Nonlinear Dynamics 104(1), 297-331, (2021) wersja PDF

   Przegląd 17 różnych iteracji, które można wykorzystać, można znaleźć tutaj:

   - Gdawiec, K., Kotarski, W.: Polynomiography for the Polynomial Infinity Norm via Kalantari's Formula and Nonstandard Iterations. Applied Mathematics and Computation 307, 17-30, (2017) wersja PDF

Algorytm 4: Renderowanie wielomianografu.

Dane: $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – wielomian; $A \subset \mathbb{C}$ – obszar; M – liczba iteracji; $I_v : \mathbb{C} \to \mathbb{C}$ – iteracja z parametrami v; $C_u : \mathbb{C} \times \mathbb{C} \to \{true, false\}$ – test zbieżności; $colours[0..k]$ – mapa kolorów.

Wynik: Wielomianograf w obszarze A.

1 **for** $z_0 \in A$ **do**
2 $\quad [n, z]$ = iteratePoint(z_0, p, I_v, C_u, M)
3 $\quad$ Pokoloruj z_0 używając n, z i mapy kolorów $colours$

Algorytm 5: iteratePoint

Dane: $z_0 \in \mathbb{C}$ – punkt; $p \in \mathbb{C}[Z]$, $\deg p \geq 2$ – wielomian; $I_v : \mathbb{C} \to \mathbb{C}$ – iteracja z parametrami v; $C_u : \mathbb{C} \times \mathbb{C} \to \{true, false\}$ – test zbieżności; M – liczba iteracji.

Wynik: Numer iteracji i ostatni obliczony punkt.

```
1 iteratePoint(z_0, p, I_v, C_u, M)
2     n = 0
3     while n < M do
4         z_{n+1} = I_v(z_n)
5         if C_u(z_n, z_{n+1}) = true then
6             break
7         n = n + 1
8     return [n, z_{n+1}]
```

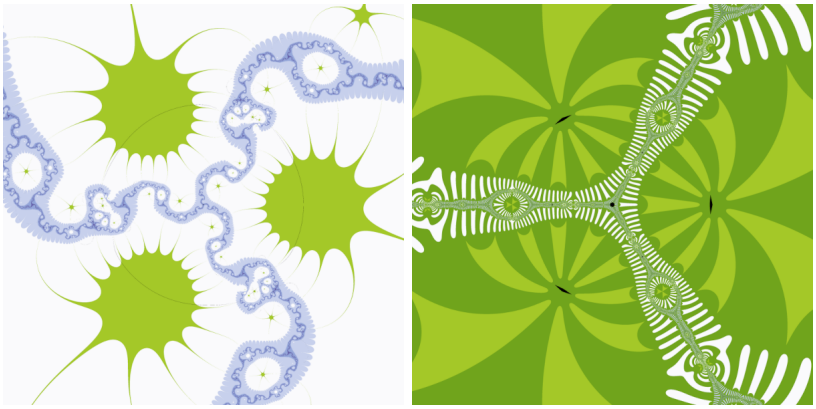
Przykład 1: wielomianografy dla metody Halleya, $p(z) = z^3 - 1$, $A = [-2, 2]^2$, $M = 30$, $\varepsilon = 0.001$, mapy kolorów z pliku *0408_093-s.map* oraz iteracji Picard–S z parametrami:

- $\alpha = 0.5 + 1.5\mathbf{i}$, $\beta = 0.8$ i testem zbieżności

$$||z_{n+1}|^2 - |z_n|^2| < \varepsilon,$$

- $\alpha = 0.1$, $\beta = 0.9$ i testem zbieżności

$$|0.01(z_{n+1} - z_n)| + |0.029|z_{n+1}|^2 - 0.03|z_n|^2| < \varepsilon.$$



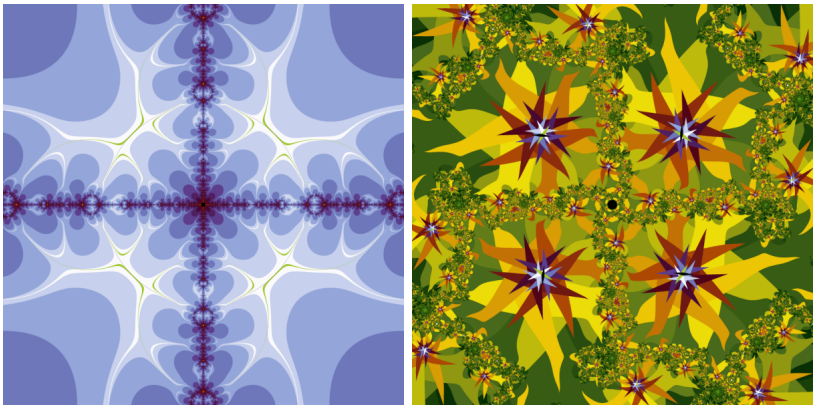
Przykład 2: wielomianografy dla metody Halleya, $p(z) = z^4 + 1$, $A = [-2, 2]^2$, $M = 30$, $\varepsilon = 0.001$, mapy kolorów z pliku *0408_093-s.map* oraz iteracji Ishikawy z parametrami:

- $\alpha = 0.9$, $\beta = 0.1$ i testem zbieżności

$$||z_{n+1}|^2 - |z_n|^2| < \varepsilon,$$

- $\alpha = 0.6 - 0.75\mathbf{i}$, $\beta = 0.8$ i testem zbieżności

$$|0.01(z_{n+1} - z_n)| + |0.029|z_{n+1}|^2 - 0.03|z_n|^2| < \varepsilon.$$



3. Zaimplementować wielomianografię wielokrokową (Algorytm 6).

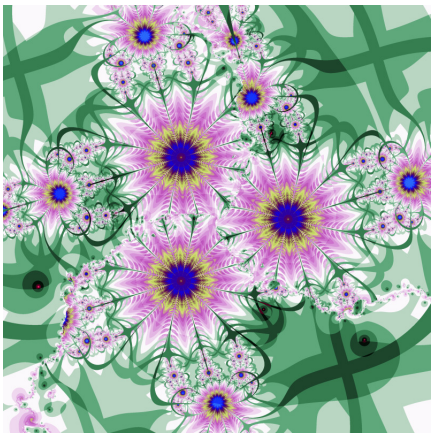
Algorytm 6: Renderowanie wielomianografu wielokrokowego.

Dane: $A \subset \mathbb{C}$ – obszar; $\{p_1, p_2, \ldots, p_N\}$ – wielomiany; $\{I_{v_1}, I_{v_2}, \ldots, I_{v_N}\}$ – iteracje;
$\{M_1, M_2, \ldots, M_N\}$ – liczby iteracji dla poszczególnych kroków;
$\{C_{u_1}, C_{u_2}, \ldots, C_{u_N}\}$ – testy zbieżności dla poszczególnych kroków;
$\{f_1, f_2, \ldots, f_N\}$ – transformacje obszaru dla poszczególnych kroków;
$colours[0..k]$ – mapa kolorów.

Wynik: Wielomianograf wielokrokowy dla obszaru A.

```
for z_0 ∈ A do
    m = 0
    z = z_0
    for i = 1, 2, ..., N do
        [n, u] = iteratePoint(z, p_i, I_{v_i}, C_{u_i}, M_i)
        m = m + n
        z = f_i(u − z)
    Wyznacz kolor dla z_0 używając m i mapy kolorów colours
```

Wielomianograf wielokrokowy:



został wygenerowany w obszarze $[-3,3]^2$ używając mapy kolorów *jutemap.map*, $\varepsilon = 0.001$ oraz następujących parametrów:

- pierwszy krok: metoda Helleya, $p_1(z) = z^3 - 1$, $M_1 = 20$, iteracja Noora z $\alpha = 0.8 - 0.4\mathbf{i}$, $\beta = 0.2$, $\gamma = 0.9 - 0.5\mathbf{i}$, $f_1(z) = z/(0.75 + 2.5\mathbf{i})$ i testu zbieżności:

$$|z_{n+1} - z_n| < \varepsilon,$$

- drugi krok: metoda Newtona, $p_2(z) = z^4 + 1$, $M_2 = 20$, iteracja Ishikawy z $\alpha = 0.7$, $\beta = 0.7$, $f_2(z) = z/(0.75 + 2.5\mathbf{i})$ i testu zbieżności:

$$|0.04\Re(z_{n+1} - z_n)| < \varepsilon \vee |0.05\Im(z_{n+1} - z_n)| < \varepsilon,$$

  gdzie $\Re(z)$, $\Im(z)$ oznaczają odpowiednio część rzeczywistą i urojoną liczby z.