Fractal methods in computer graphics

Assignments set 2

1. Implement the deterministic algorithm of generating circle inversion fractal (Algorithm 1).

Algorithm 1: Deterministic inversion algorithm.

Input: $C_1, \ldots, C_k$ – inversion circles, $S_1, \ldots, S_m$ – starting circles, each intersects some of the C_i and is orthogonal to all the C_i they intersect, n – the number of iterations.

Output: F – circle inversion fractal

```
1  F = {S_1, ..., S_m}
2  T_1 = ∅
3  for i = 1, ..., k do
4      for j = 1, ..., m do
5          if S_j does not intersect C_i then
6              C = I_{C_i}(S_j)
7              T_1 = T_1 ∪ C
8  F = F ∪ T_1
9  for l = 1, ..., n do
10     T_2 = ∅
11     for i = 1, ..., k do
12         for j = 1, ..., |T_1| do
13             S = j-th circle from T_1
14             if C_i is not orthogonal to S and i ≠ q, where q is the number of the inversion circle
                used in the previous iteration to obtain S then
15                 C = I_{C_i}(S)
16                 T_2 = T_2 ∪ C
17     F = F ∪ T_2
18     T_1 = T_2
19 Plot F
```

In order to realize the deterministic method we will need two test: test for intersection of two circles, test checking whether two circles are orthogonal. Let us assume that $c_1, c_2 \in \mathbb{R}^2$ are the centres of two circles, and r_1 and r_2 are their radii, respectively.

In the first test, we compare the distance between the centres of the circles and their radii (school mathematics).

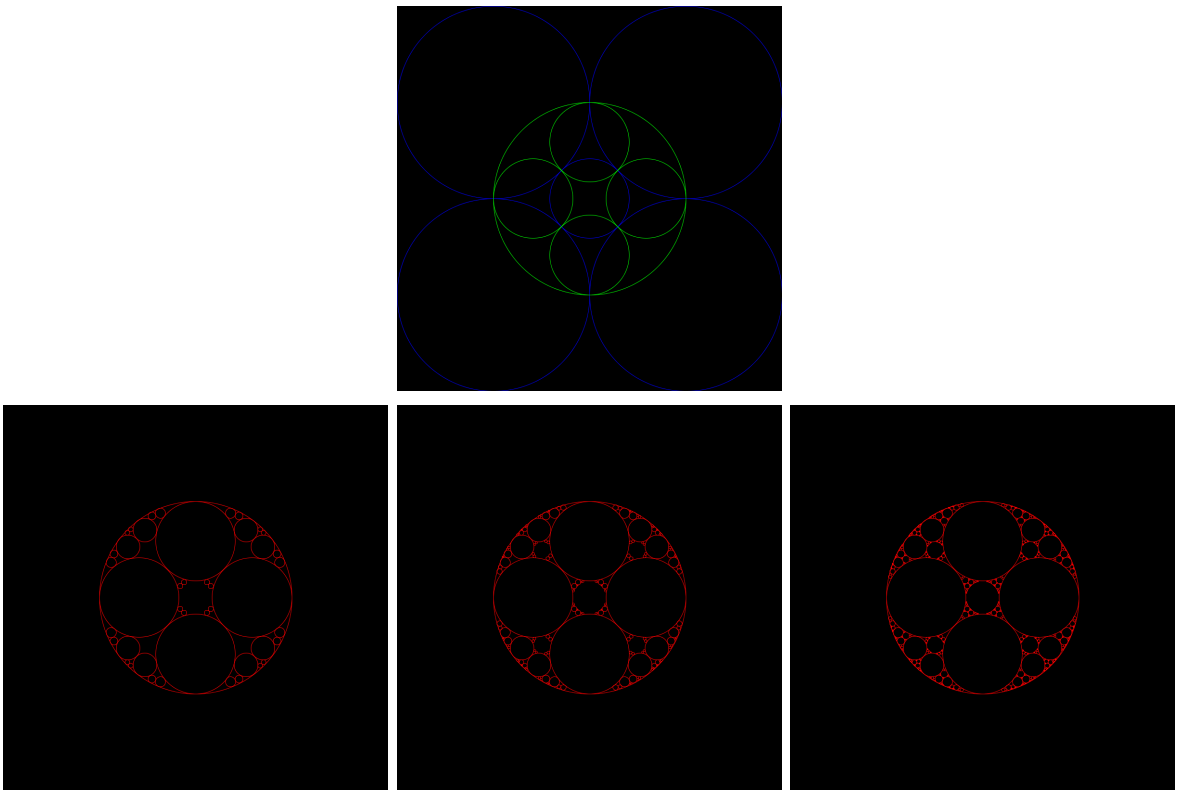
In the case of the second test, the circles are orthogonal if the following condition is satisfied:

$$r_1^2 + r_2^2 = d^2,$$

where d is the Euclidean distance between c_1 and c_2.

An example of the first three iterations in the generation of a circle inversion fractal (blue – inversion circles, green – the starting circles, red – the circles forming the circle

inversion fractal):



The parameters of the inversion circles from the example: $c_1 = (200, 200)$, $r_1 = 200$, $c_2 = (600, 200)$, $r_2 = 200$, $c_3 = (200, 600)$, $r_3 = 200$, $c_4 = (600, 600)$, $r_4 = 200$, $c_5 = (400, 400)$, $r_5 = 82.85$.

The parameters of the starting circles from the example: $c_1 = (400, 400)$, $r_1 = 200$, $c_2 = (400, 282.85)$, $r_2 = 82.85$, $c_3 = (400, 517.15)$, $r_3 = 82.85$, $c_4 = (282.85, 400)$, $r_4 = 82.85$, $c_5 = (517.15, 400)$, $r_5 = 82.85$.

2. Implement the generation algorithm of the star-shaped set inversion fractal (Algorithm 2). In the program we should be able to generate fractal which is given by circles and star-shaped polygons. As the iteration one should use the Mann iteration.

   In the implementation we will need to find the intersection point of a ray with the boundary of the star-shaped set. Let us assume that the ray is given by the formula $r(t) = o + t(p - o)$, where $t \in [0, \infty)$, o – inversion centre, p – the point defining the ray.

   In the case of a circle with the centre c and radius r after including the formula for the ray into the circle equation we get:

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

   where $\cdot$ is the dot product. This is a quadratic equation with the variable t. By solving this equation we get: no solution (there is no intersection), one solution (the ray is tangent to the circle), or two solutions (because we consider a ray, so the solution which we are looking for is the positive solution). After finding the solution t_*, we put its value into the ray's equation $r(t_*)$ obtaining the searched intersection point.

Algorithm 2: Extended random inversion algorithm with colouring

Input: $S_1, \ldots, S_k$ – star-shaped sets with chosen centres of inversion, $c_1, \ldots, c_k$ – colours of the transformations, p_0 – starting point external to $S_1, \ldots, S_k$, $n > 20$ – number of iterations, P_v – iteration with parameters v, W, H – image dimensions, $\gamma \in \mathbb{R}_+$

Output: Image I with an approximation of a star-shaped set inversion fractal

for $(x, y) \in \{0, 1, \ldots, W-1\} \times \{0, 1, \ldots, H-1\}$ **do**
    $I(x, y) =$ black
    $\mathcal{H}(x, y) = 0$
$c =$ a random colour
$j =$ a random number from $\{1, \ldots, k\}$
$p = P_v(I_{S_j}, p_0)$
for $i = 2$ **to** n **do**
    $l =$ a random number from $\{1, \ldots, k\}$
    while $j = l$ **or** $inSet(S_l, p)$ **do**
        $l =$ a random number from $\{1, \ldots, k\}$
    $j = l$
    $p = P_v(I_{S_j}, p)$
    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**
        $I(x, y) = \left(\frac{\log_2(1 + \mathcal{H}(x,y))}{\log_2(1 + m_{\mathcal{H}})} \right)^{1/\gamma} I(x, y)$

In the case of star-shaped polygon, we check intersection of the ray with every edge of the polygon. Because the polygon is star-shaped, so there is only one such point. Therefore, when we find the intersection point we do not need to test the next edges of the polygon. Because the point p defining the ray lies outside the star-shaped set, so to check the intersection of a ray with an edge we can use Algorithm 3.

Algorithm 3: Ray–edge intersection test.

Input: p_0, p_1 – edge's endpoints; o – ray's origin; p – the point defining the direction of the ray.

Output: The intersection point or *null* if there is no intersection.

```
1  d_r = p − o
2  d_s = p_1 − p_0
3  q = p_0 − o
4  d = d_ry d_sx − d_rx d_sy
5  if |d| < 0.0001 then
6      return null
7  t = (1/d)(d_sx a_y − d_sy a_x)
8  s = (1/d)(d_rx a_y − d_ry a_x)
9  if t ≥ 0 ∧ s ∈ [0, 1] then
10     return o + t d_r
11 return null
```

To a full implementation of Algorithm 2 we need also a test that checks whether a given point p is inside the star-shaped set. In the case of the circle, we can use school mathematics, i.e., check a condition that results from circle's equation. For a star-shaped polygon, the situation is more complex. We can use algorithm that is presented in Algorithm 4.

Exemplary star-shaped sets defining some star-shaped set inversion fractals are in the *inv_fractals.zip* archive. The format of data included in the files is the following. In the first line we have a single number k which is the number of the star-shaped sets. In the next k lines we have definitions of the sets. Each line starts with a single letter: C – circle, P – polygon. In the case of the circle the next two numbers are the inversion's centre coordinates, and the next three numbers are the centre's coordinates and the radius. In the case of the polygon, the first two numbers are the inversion's centre coordinates. Then, we have the number of vertices, and next each pair of numbers are the vertex coordinates.

Example of a star-shaped set inversion fractal (*various_03.sif*) for various values of the α parameter in the Mann iteration. The first image presents the star-shaped sets, and the next ones present fractals for α: 1.0, 0.9, 0.8, 0.7, 0.6.

Algorithm 4: Point inside a star-shaped polygon test.

Input: $v_0, v_1, \ldots, v_n$ – polygon's vertices; *orient* – vertices orientation (*true* – counterclockwise orientation, $false$ – clockwise orientation); p – the point that we test.

Output: *true* – the point is inside the polygon; $flase$ – the point is outside of the polygon.

```
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
```