



Chaos game

Wydanie 1.0

Jacek Siciarek

November 10 2016

Spis treści

1	Spis treści	1
1.1	Chaos game czyli nauka panowania nad chaosem, dla programistów, w ramach spędzania wolnego czasu	1

Spis treści

Chaos game czyli nauka panowania nad chaosem, dla programistów, w ramach spędzania wolnego czasu

Chaos game (gra w chaos) dla niewtajemniczonego brzmi dość zły i przywodzi na myśl gry typu FPP, gdzie do tłumów kosmicznych najeźdźców strzela się z różnych strzelających urządzeń, najczęściej dość dużego kalibru.

Prawda nie jest aż tak atrakcyjna dla fanów e-sportu, aczkolwiek może sprawić, że przez chwilę opuścimy broń i zastanowimy się nad istotą chaosu (oczywiście tego policzalnego). Bo prawdą jest, że wiele rzeczy, które nazywamy *chaosem* oznacza coś czego nie rozumiemy i nie potrafimy wykorzystać, tak było ze statystyką, medycyną czy pogodą, jednak po latach badań nauczyliśmy się wydobywać z chaosu wiedzę.

Chaos game może nie jest aż tak użytecznym elementem teorii chaosu jak Atraktor Lorenza, czy bifurkacja, ale pozwoli w sposób widoczny, łatwy i przyjemny dotknąć natury zagadnienia.

O co chodzi z tym całym chaosem

Termin *Chaos game* został pierwszy raz podany przez Michaela F. Barnsleya (tak, tego od paprotki) w 1988 r. w słynnej publikacji *Fractals Everywhere* i dotyczył prostego algorytmu, który można opisać następująco:

Warunki początkowe

- W układzie współrzędnych wyznaczyć zbiór n wierzchołków wielokąta (najlepiej foremnego, Barnsley użył trójkąta równobocznego).
- Losowo wybrać pierwszy *punkt aktywny* (może być dowolnym punktem na płaszczyźnie, wewnętrz lub na zewnątrz wielokąta, może być nawet jednym z jego wierzchołków, Barnsley użył punktu wewnętrz).

Algorytm

- Narysuj *punkt aktywny*.

- Wylosuj *wierzchołek wielokąta* z uwzględnieniem tabeli ograniczeń q (w pierwotnej wersji nie było ograniczeń).
- Wyznacz *nowy punkt* należący do odcinka łączącego *punkt aktywny* z wylosowanym wierzchołkiem wielokąta znajdujący się odległości r od wylosowanego wierzchołka wielokąta, gdzie $0 < r < 1$ (w pierwotnej wersji $r = 0.5$).
- Oznacz *nowy punkt* jako *punkt aktywny*.

Algorytm powtarzamy wyznaczoną ilość powtórzeń *times*. W literaturze stosuje się zazwyczaj $times \approx 10000$, jako dość wydajny kompromis pomiędzy efektem wizualnym a łącznym czasem wykonania algorytmu.

W wyniku manipulacji parametrami n, r, q oraz *times*, podobno, można osiągnąć interesujące wizualnie wyniki, co mam nadzieję udowodnić w dalszej części.

Dla wygody zakładamy, że

- wszystkie wielokąty będą foremne,
- pierwszy wierzchołek każdego wielokąta będzie umiejscowiony na górze i będzie posiadał takie same współrzędne,
- pierwszy *punkt aktywny* będzie miał współrzędne $(0, 0)$,
- wartości współrzędnych wierzchołków, będą większe od zera (jak to w grafice komputerowej),
- zaś domyślna ilość powtórzeń *times* = 100000.

Oczywiście zachęcam do eksperymentowania z wartościami tych ustawień, jednak uprzedzam, nie mają zbyt dużego wpływu na osiągane wyniki.

Do dzieła

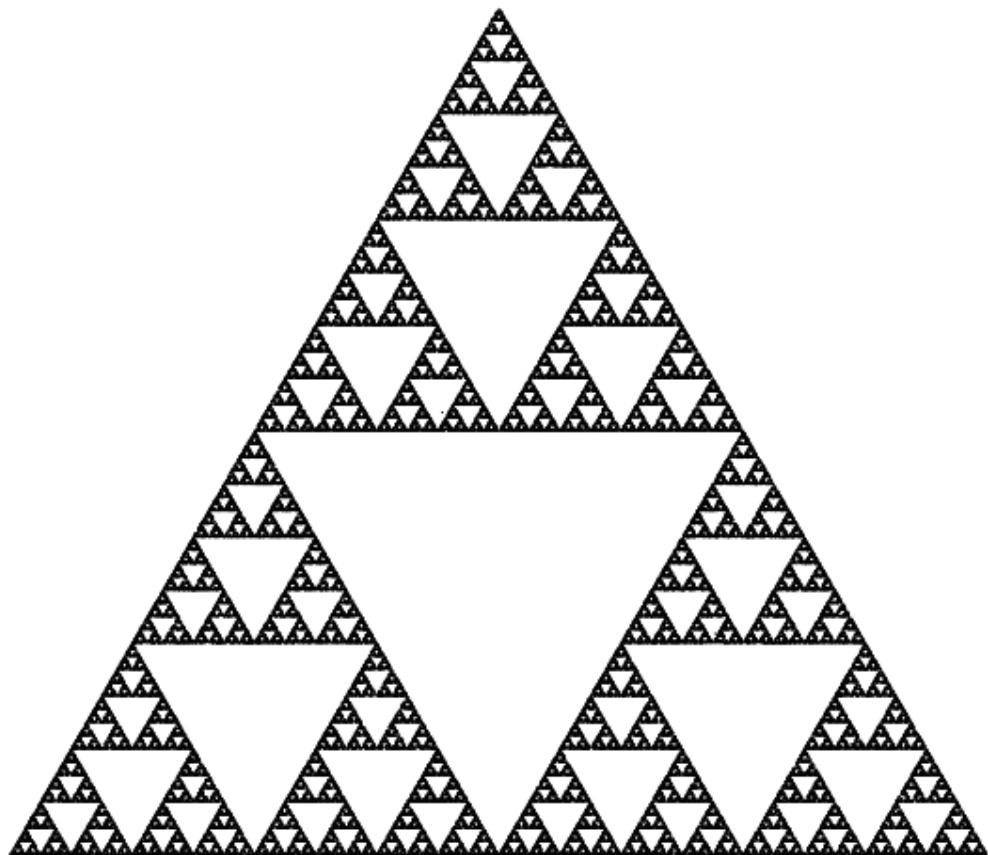
Zabawę zaczniemy od *trójkąta równobocznego*, a wyznaczone przez nas parametry będą przedstawiały się następująco:

$$n = 3, r = 0.5, q = ()$$

Co oznacza:

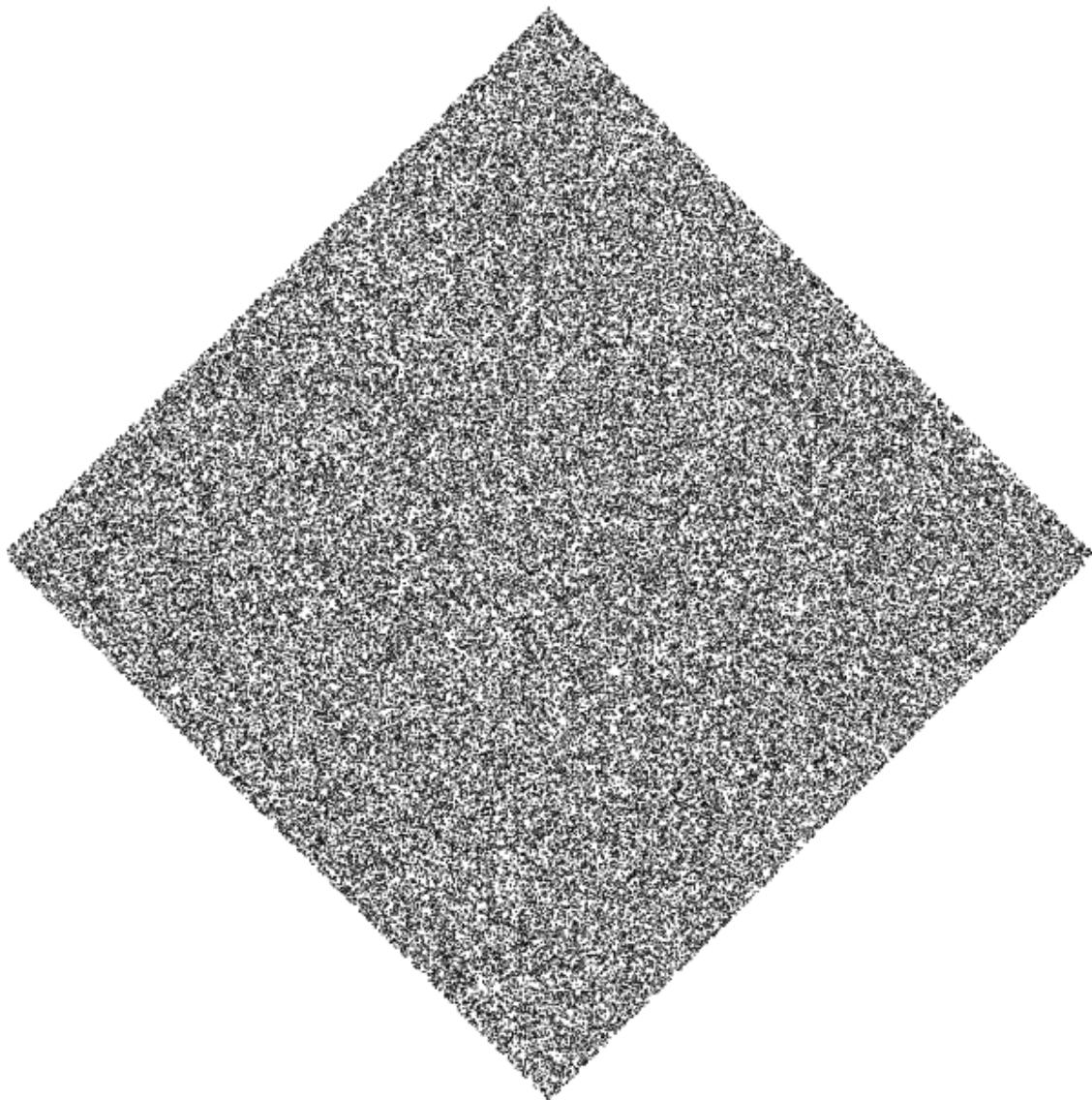
- wielokąt będzie miał trzy wierzchołki,
- nowy punkt pojawi się w połowie odległości między *punktem aktywnym* a *wylosowanym wierzchołkiem*,
- ponieważ tabela ograniczeń q jest pusta, wierzchołki losowane są bez żadnych dodatkowych ograniczeń.

Oto co otrzymujemy.



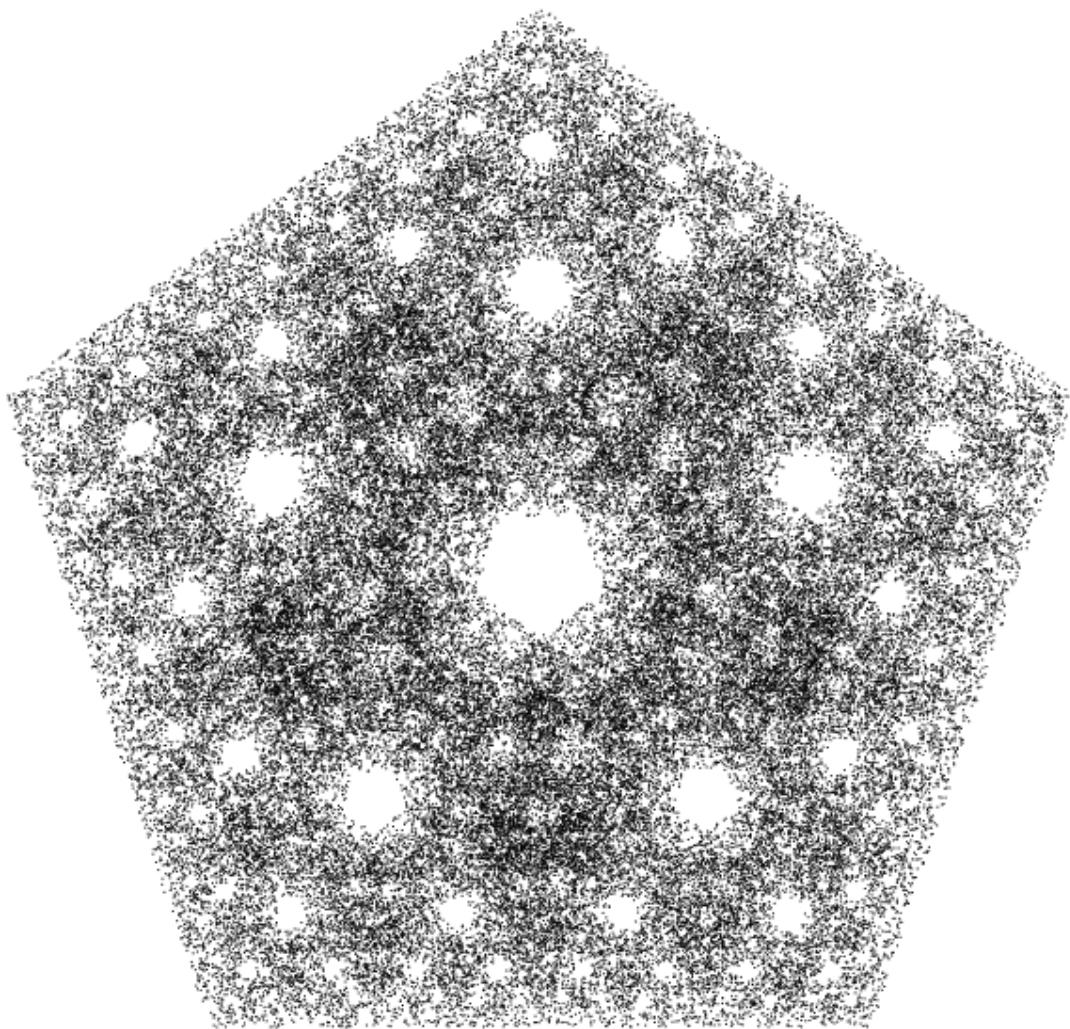
Pojawił się Trójkąt Sierpińskiego, całkiem atrakcyjny wizualnie układ, mimo losowości wybierania wierzchołków. Świeście, spróbujmy teraz z większą ilością wierzchołków, następny w kolejce *kwadrat*, opisujemy go następująco:

$$n = 4, r = 0.5, q = ()$$

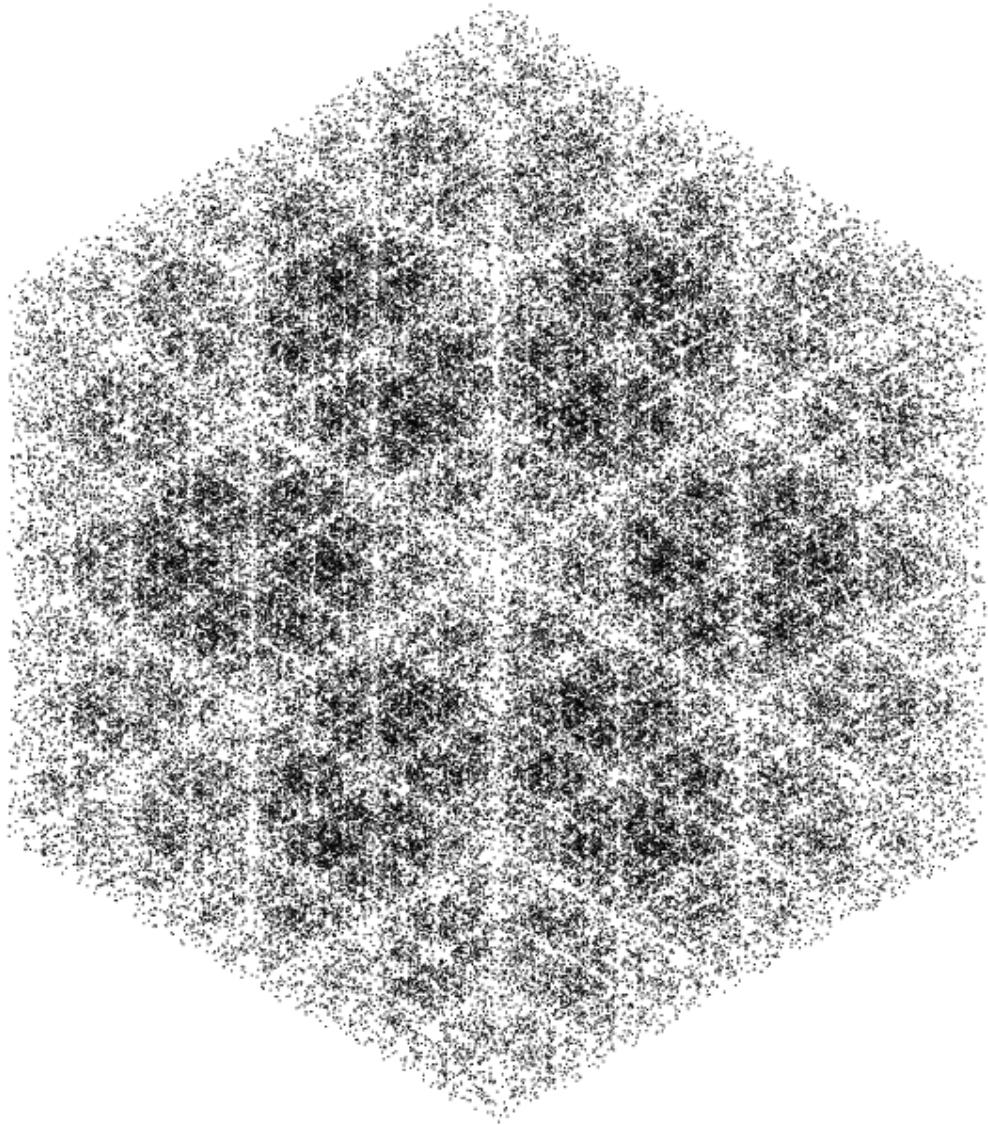


Rezultat niestety nie zachwyca. Spróbujmy dla większej ilości wierzchołków, np. użyjmy *pięciokąta i sześciokąta foremnego*.

$$n = 5, r = 0.5, q = ()$$



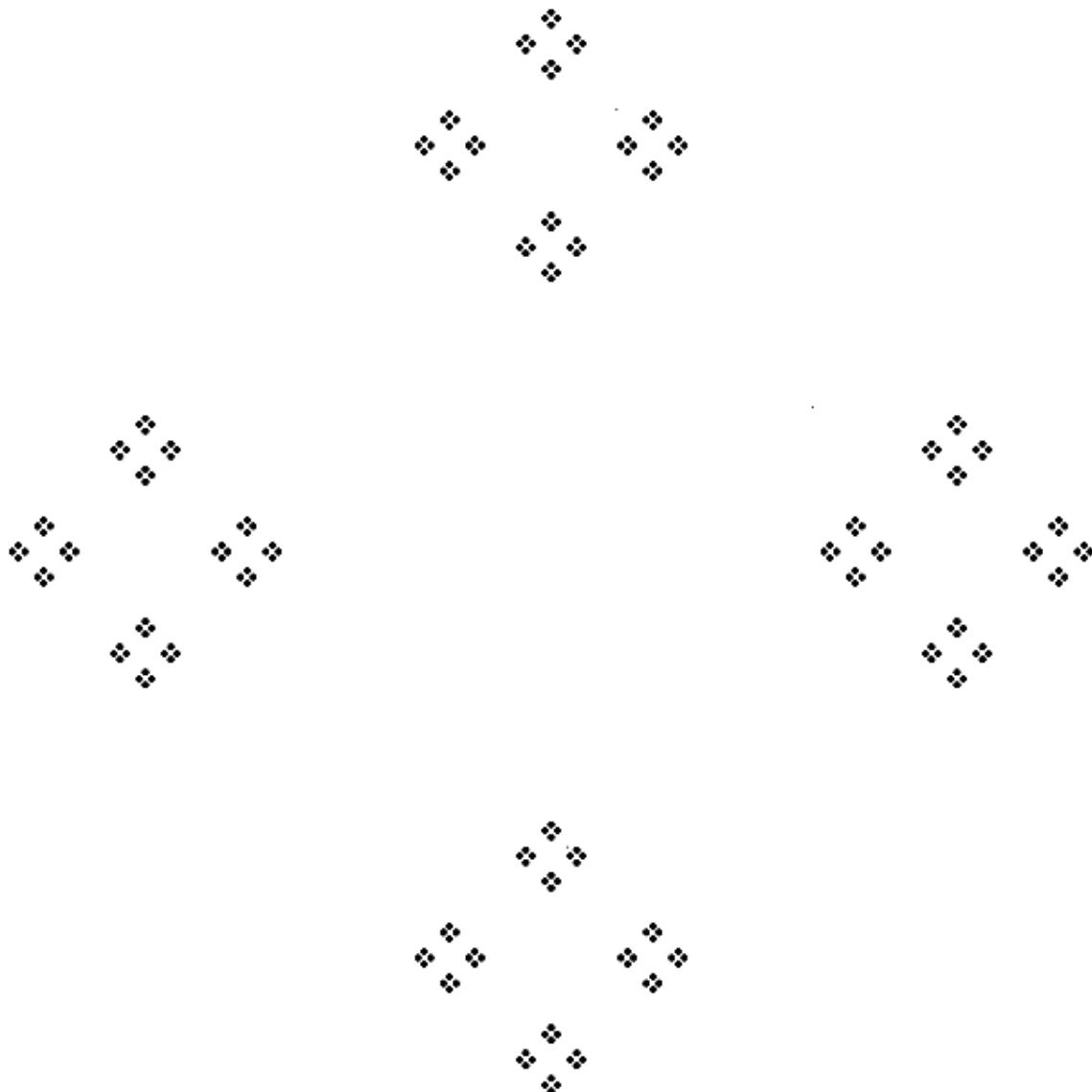
$n = 6, r = 0.5, q = ()$



Chyba nie tedy droga, chaos jak to chaos rządzi się swoimi prawami, bardzo ładnie zadziałał na trójkącie, ale zmiana ilości wierzchołków daje wyniki na poziomie szumu, jedno co widzimy, to to, że pomimo wyboru punktu startowego na zewnątrz figury, prawie wszystkie punkty wyładowały wewnątrz.

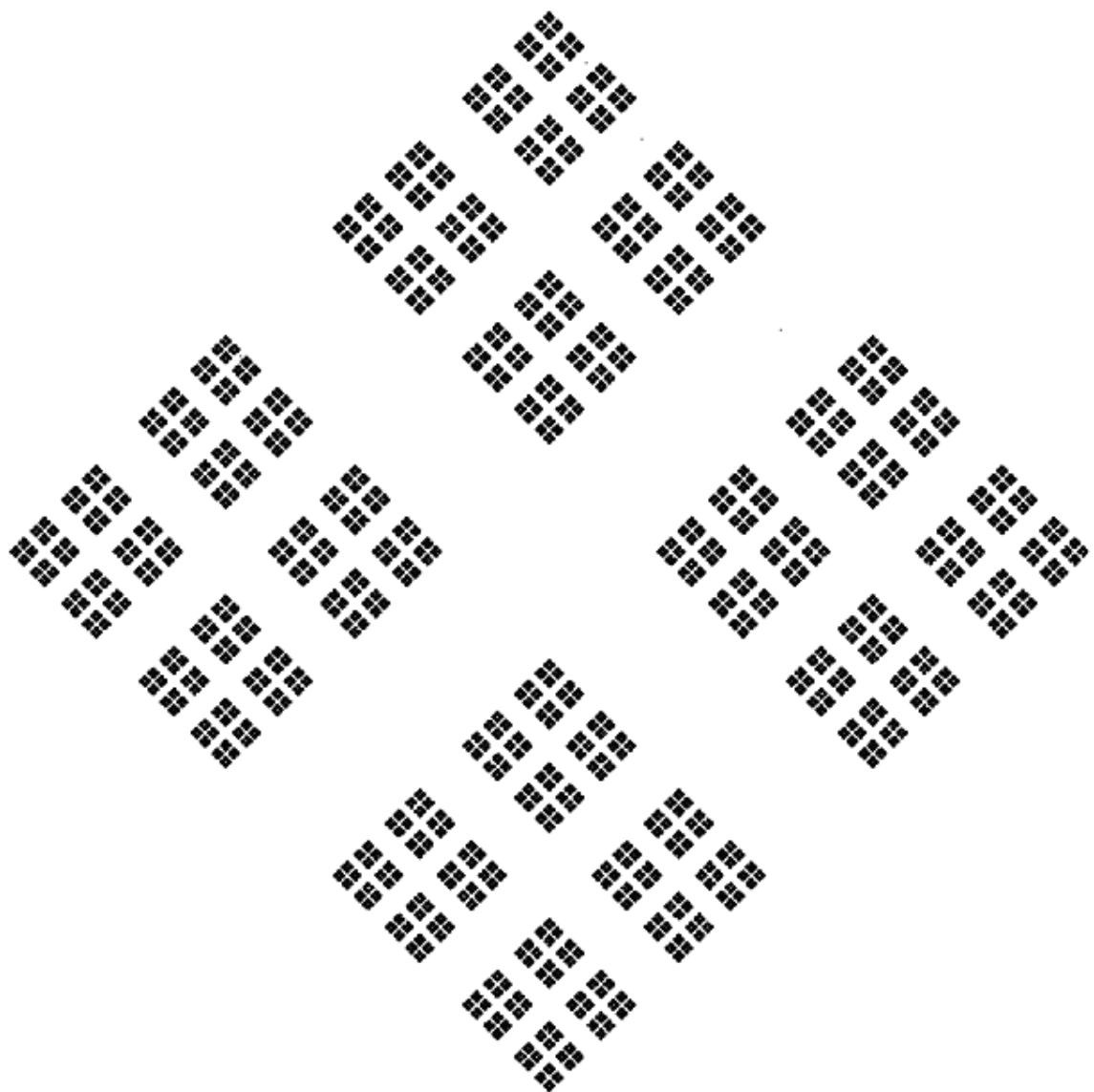
Ponieważ zmiana ilości boków bez zmiany innych parametrów nie przyniosła zbyt spektakularnych efektów wracamy do *kwadratu*, ale zmienimy parametr r o połowę.

$$n = 4, r = 0.25, q = ()$$

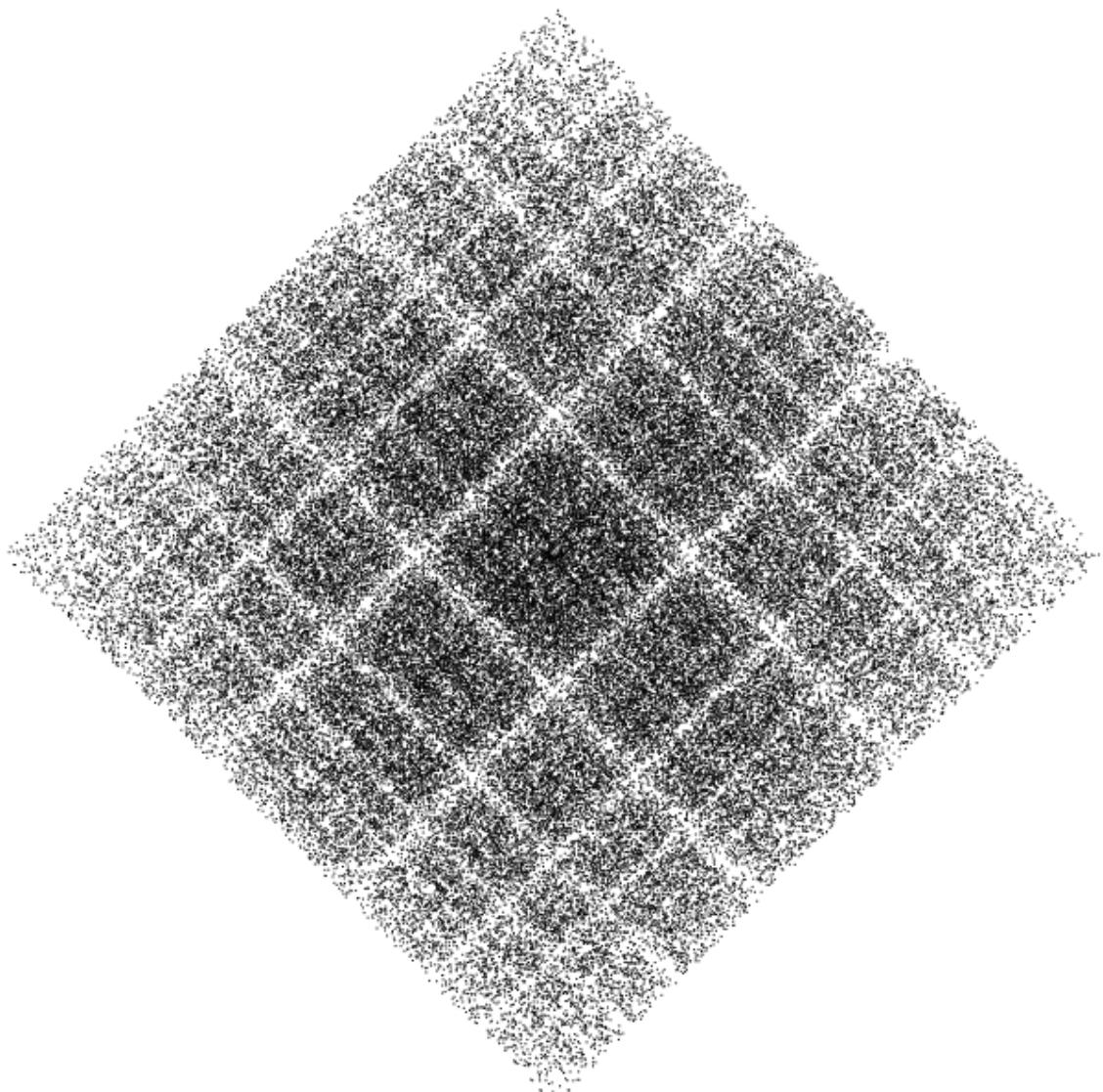


No to wygląda znacznie lepiej, duże uporządkowanie, aczkolwiek dużo punktów trafiło na niewielką powierzchnię. Ciekawe co stanie się jeżeli pozmieniamy nieco parametr r , np. nadając mu wartości odpowiednio 0.4, 0.6, 0.75 i 0.9.

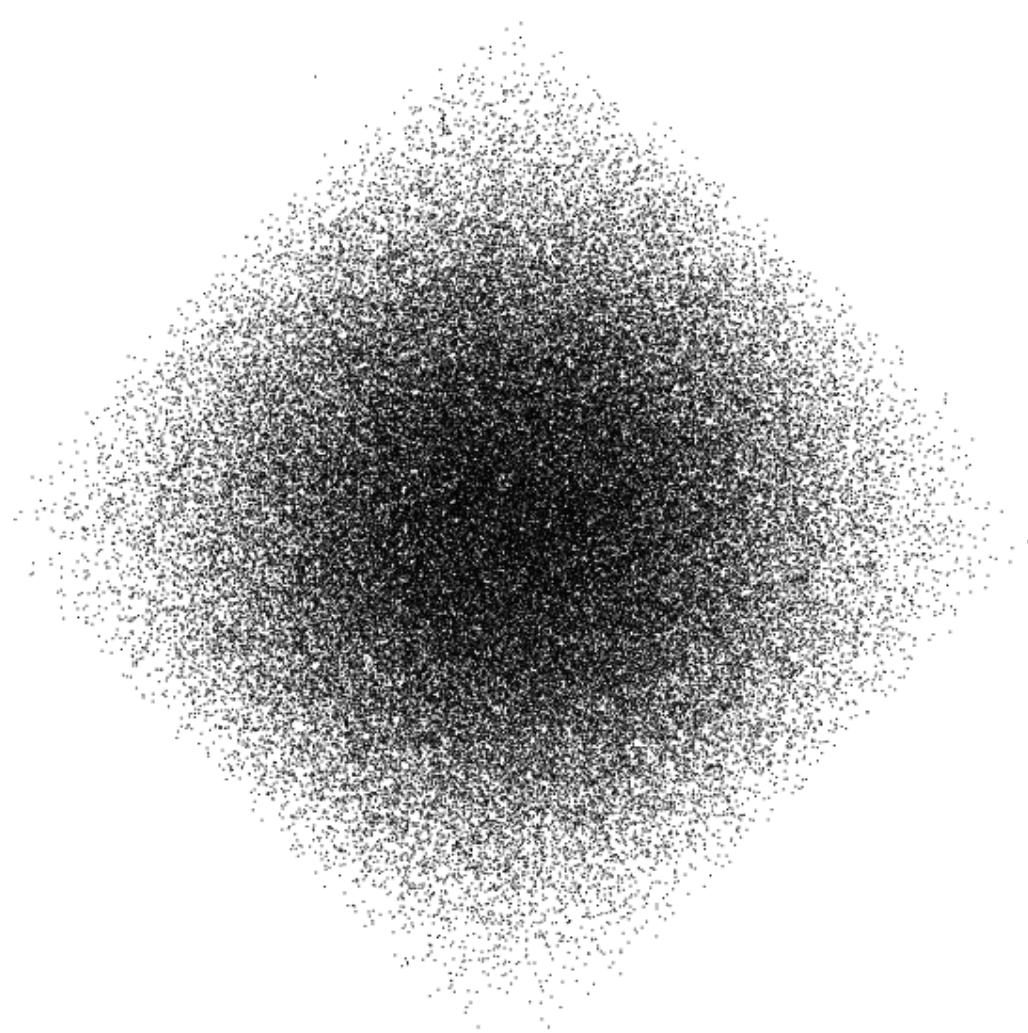
$$n = 4, r = 0.4, q = ()$$



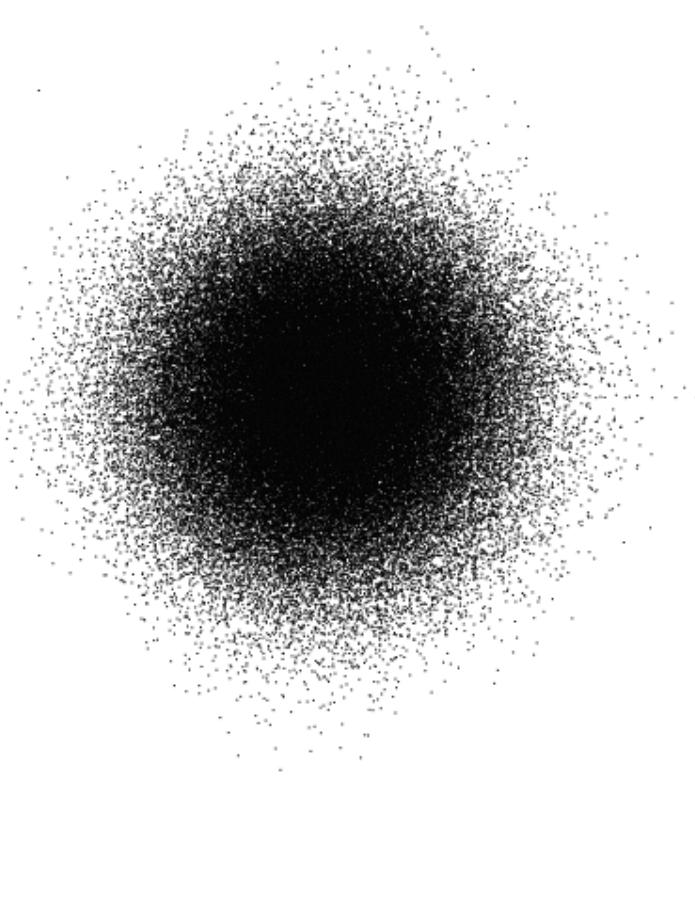
$n = 4, r = 0.6, q = ()$



$n = 4, r = 0.75, q = ()$



$n = 4, r = 0.9, q = ()$



Widać, że uporządkowanie chaosu przebiega nieregularnie i o ile dla wartości 0.25 i 0.4 wygląda obiecująco, przy 0.5 zanika, zaś dla 0.6 zdaje się wprowadzać jakieś “nitkowe” uporządkowanie o tyle dla > 0.75 wszystko zaczyna być wchłaniane przez “czarną dziurę”. Chciałoby się powiedzieć jak to w chaosie, biorąc pod uwagę pierwotne znaczenie tego słowa, jako kosmicznej pramaterii.

Więcej o tabeli ograniczeń

Dotychczas korzystaliśmy wyłącznie z pustej tabeli ograniczeń q , zanim zaczniemy tam umieszczać wartości, należy się kilka słów wyjaśnienia, z czym to się je, to znaczy, co zawiera ta tablica i jak z niej korzystać.

Wyjdziemy jednak od sposobu w jaki losowane są wierzchołki, otóż wierzchołki figury użytej do gry przechowywane są w tablicy i w wyniku losowania wybierany jest jeden z indeksów tej tablicy, liczba całkowita z przedziału $[0, n - 1]$. W tabeli q umieszczone będą *odległości* wylosowanego indeksu w stosunku do zbioru ostatnich losowań. Odległość oznacza ilość pozycji

od danego indeksu. Weźmy pod uwagę zbiór czterech punktów:

$$points = (0, 0), (200, 0), (200, 200), (0, 200)$$

jeżeli wylosowano wierzchołek o indeksie 0 (0, 0) to w odległości 0 znajduje się ten sam punkt, więc ograniczenie $q = (0)$ oznacza, ni mniej ni więcej, tylko nowo wylosowany punkt, nie może być użyty, jeżeli został wylosowany poprzednim kroku.

Odległość 1 oznacza wierzchołki znajdujące się w odległości 1 (oczywiście w pierścieniu modulo n) od wybranego wierzchołka, czyli dla podanego punktu (0, 0) będzie to (200, 0). Analogicznie odległość 2 oznacza dla tego przypadku (200, 200) a 3 (0, 200).

Powyższe nakłada podstawowe ograniczenie na dopuszczalne wartości:

Każda wartość tablicy q musi być liczbą całkowitą i należeć do zbioru $0, 1, \dots, n - 1, n$

Ciekawiej zaczyna się robić gdy tabela ograniczeń ma więcej niż jeden element, działa to mniej więcej tak, że na pozycji

- 0 jest odległość dotycząca ostatniego udanego losowania
- 1 jest odległość dotycząca przedostatniego udanego losowania
- 2 jest odległość dotycząca przedostatniego udanego losowania
- i tak dalej, i tak dalej.

Uwaga! losowanie jest uznane za udane, wtedy i tylko wtedy, gdy wylosowany indeks nie spełnia tych odległości.

W tym celu wewnętrz aplikacji wspierającej grę w chaos musi być przechowywana lista ostatnio dokonanych udanych losowań o rozmiarze przynajmniej takim jak tabela ograniczeń q . Przykład więcej niż jednej wartości w tabeli ograniczeń:

Ostatnie 2 losowania wyglądały następująco:

$$log = (0, 1)$$

Tabla ograniczeń:

$$q = (2, 1)$$

Wylosowano indeks 0 odległość od indeksu z ostatniego udanego losowania wynosi 2, warunek nie spełniony, ponawiamy losowanie. Wylosowano indeks 1 odległość od indeksu z ostatniego udanego losowania wynosi 1, ale odległość od przedostatniego wynosi 1 warunek nie spełniony, ponawiamy losowanie. Wylosowano indeks 3 odległość od indeksu z ostatniego udanego losowania wynosi 1, odległość od przedostatniego wynosi 2 warunek spełniony, losowanie udane.

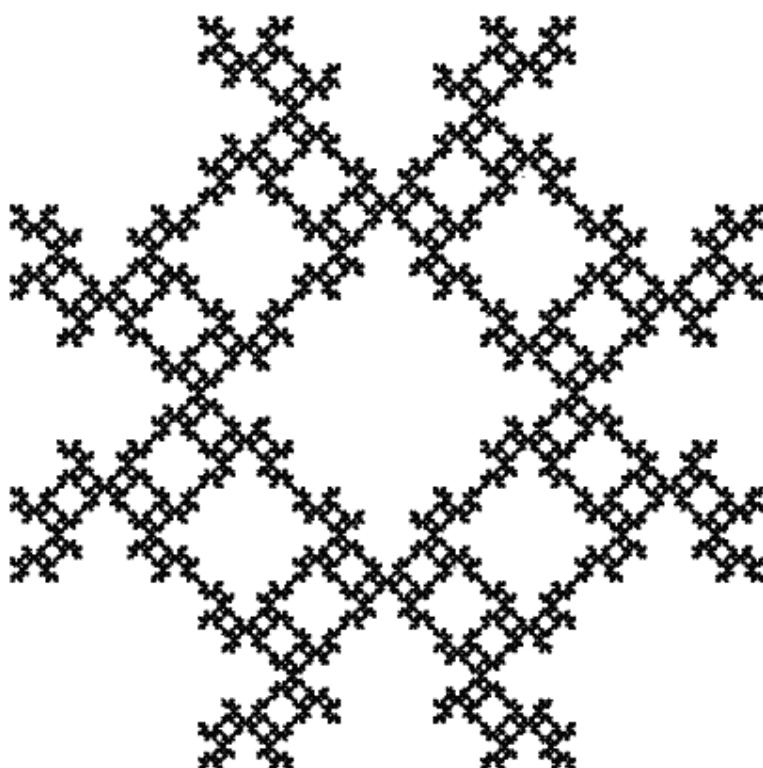
Trochę to zakręcone, ale łatwiej będzie zrozumieć gdy zobaczymy wyniki. W bieżącej publikacji zajmować się będziemy tabelami ograniczeń zawierającymi nie więcej niż 2 elementy.

Gra z tabelą ograniczeń

Postaramy się teraz założyć ograniczenia przy pomocy niepustej tablicy q .

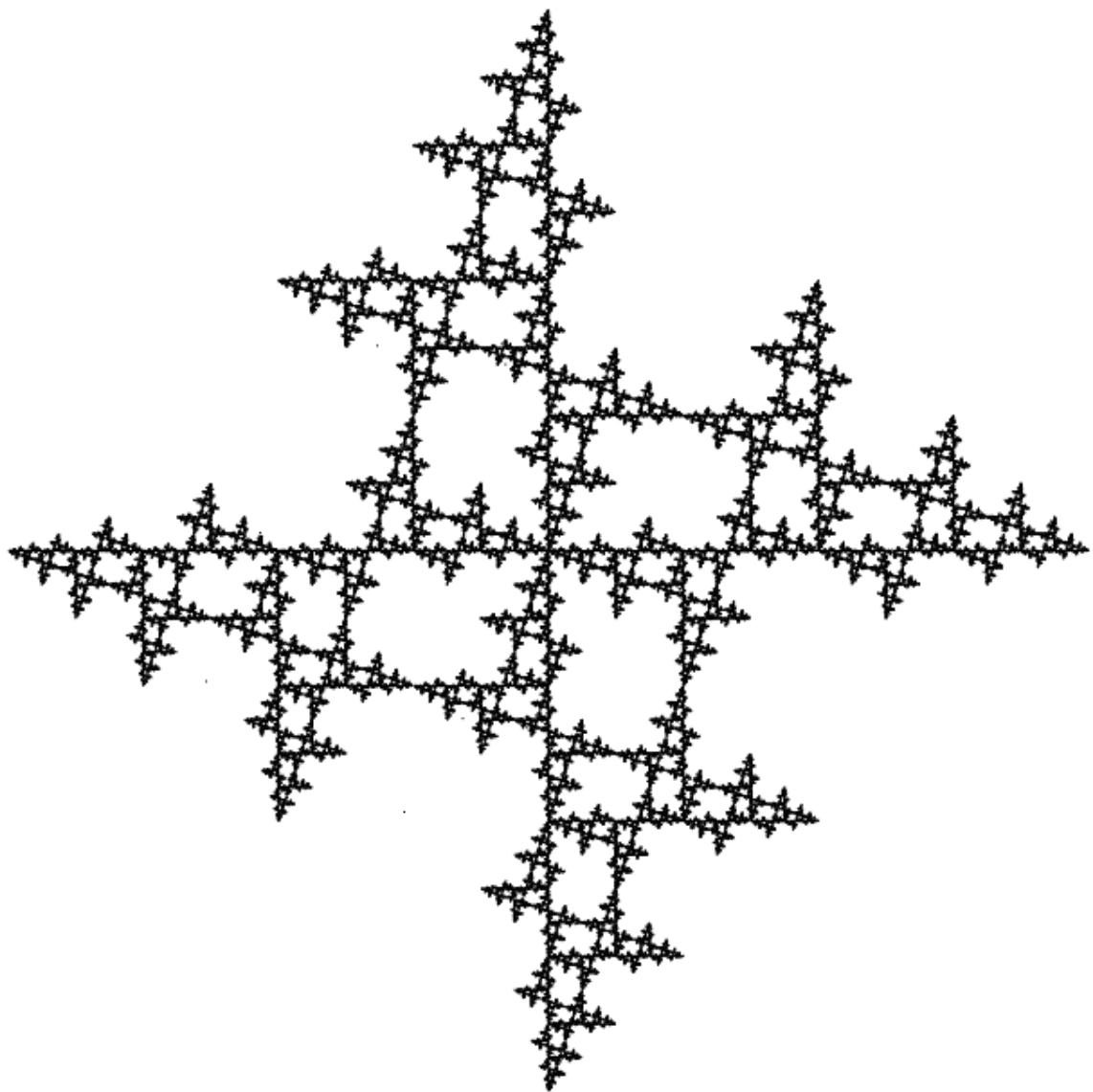
Pozostaniemy przy kwadracie $n = 4$, wartości $r = 0.5$ i spróbujemy wykonać ćwiczenie z jednoelementową tabelą ograniczeń o wartościach, odpowiednio 0, 1, 2 i 3.

$$n = 4, r = 0.5, q = (0)$$

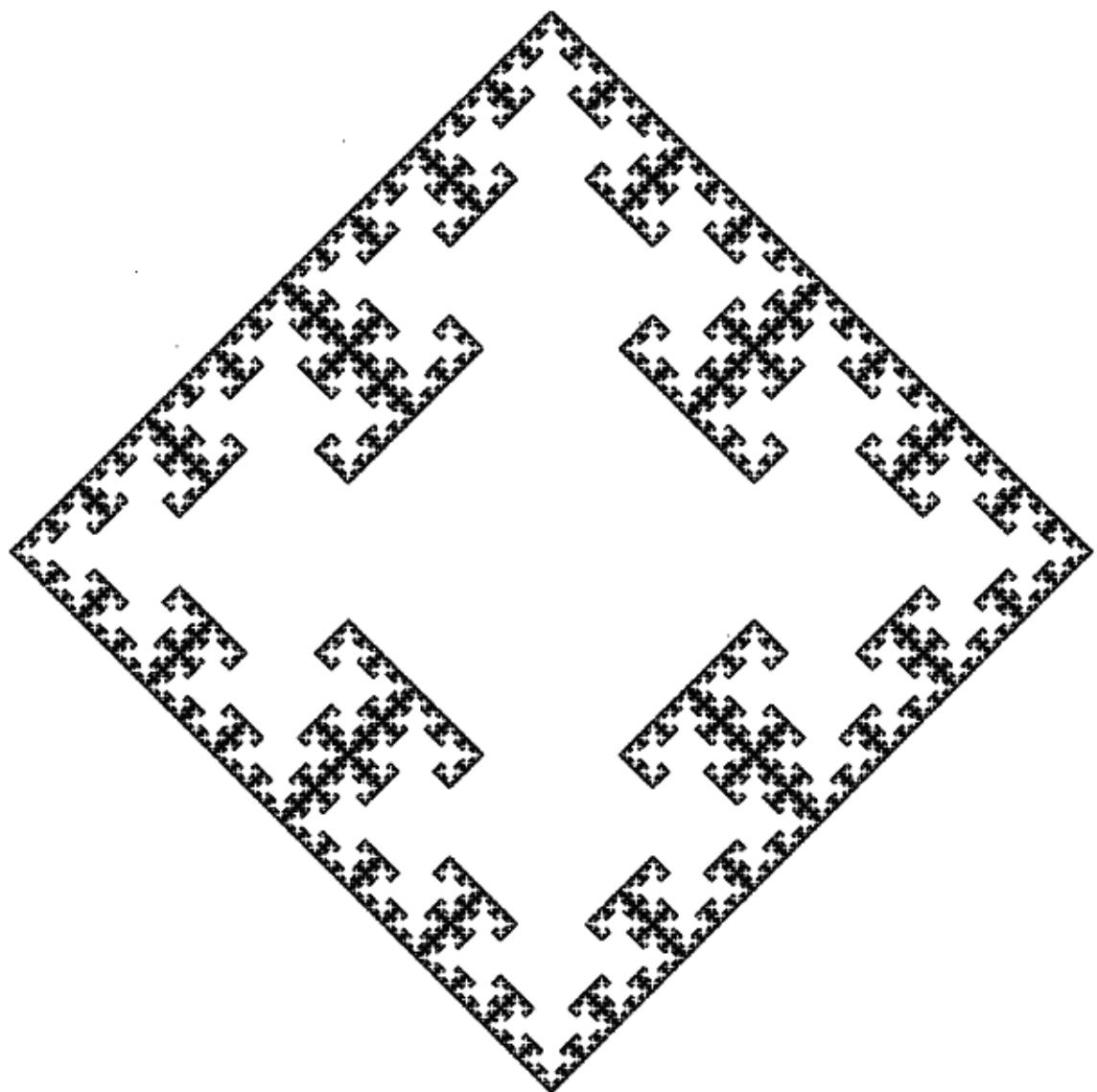


No, to można nazwać sukcesem, elegancki fraktal przypominający mapę albo jakieś mury obronne. Próbowejmy kolejnych wartości.

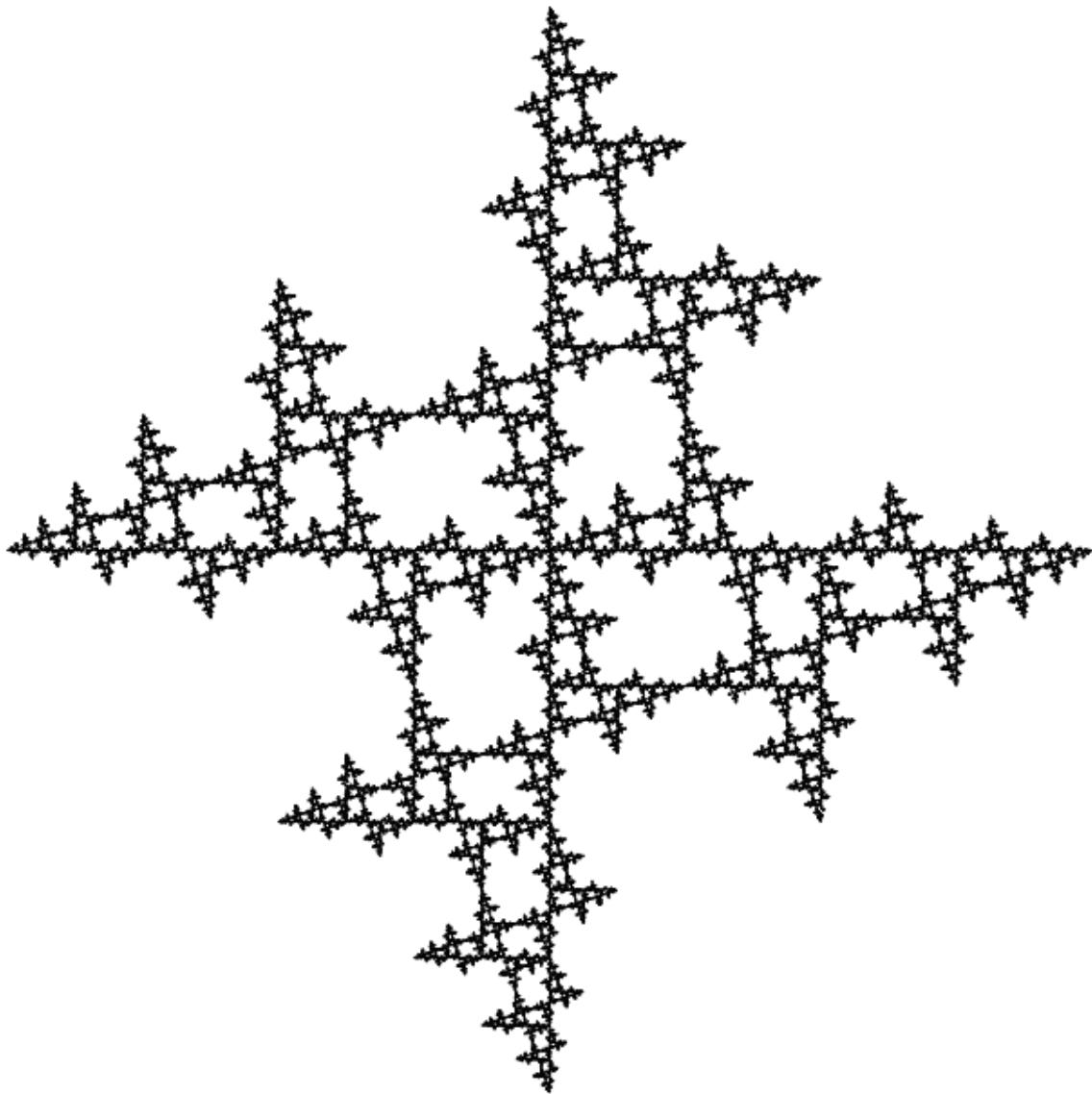
$$n = 4, r = 0.5, q = (1)$$



$n = 4, r = 0.5, q = (2)$



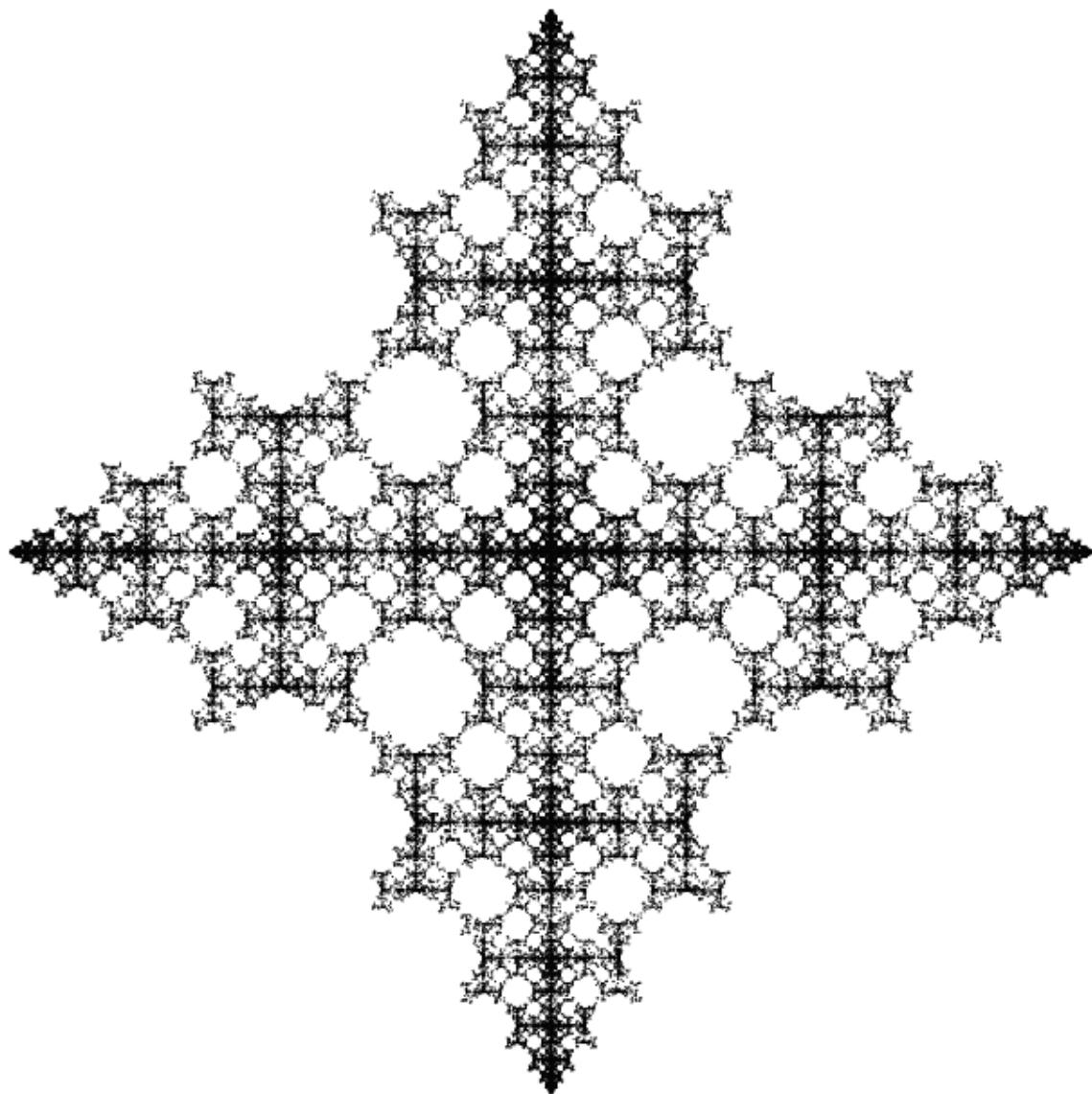
$n = 4, r = 0.5, q = (3)$



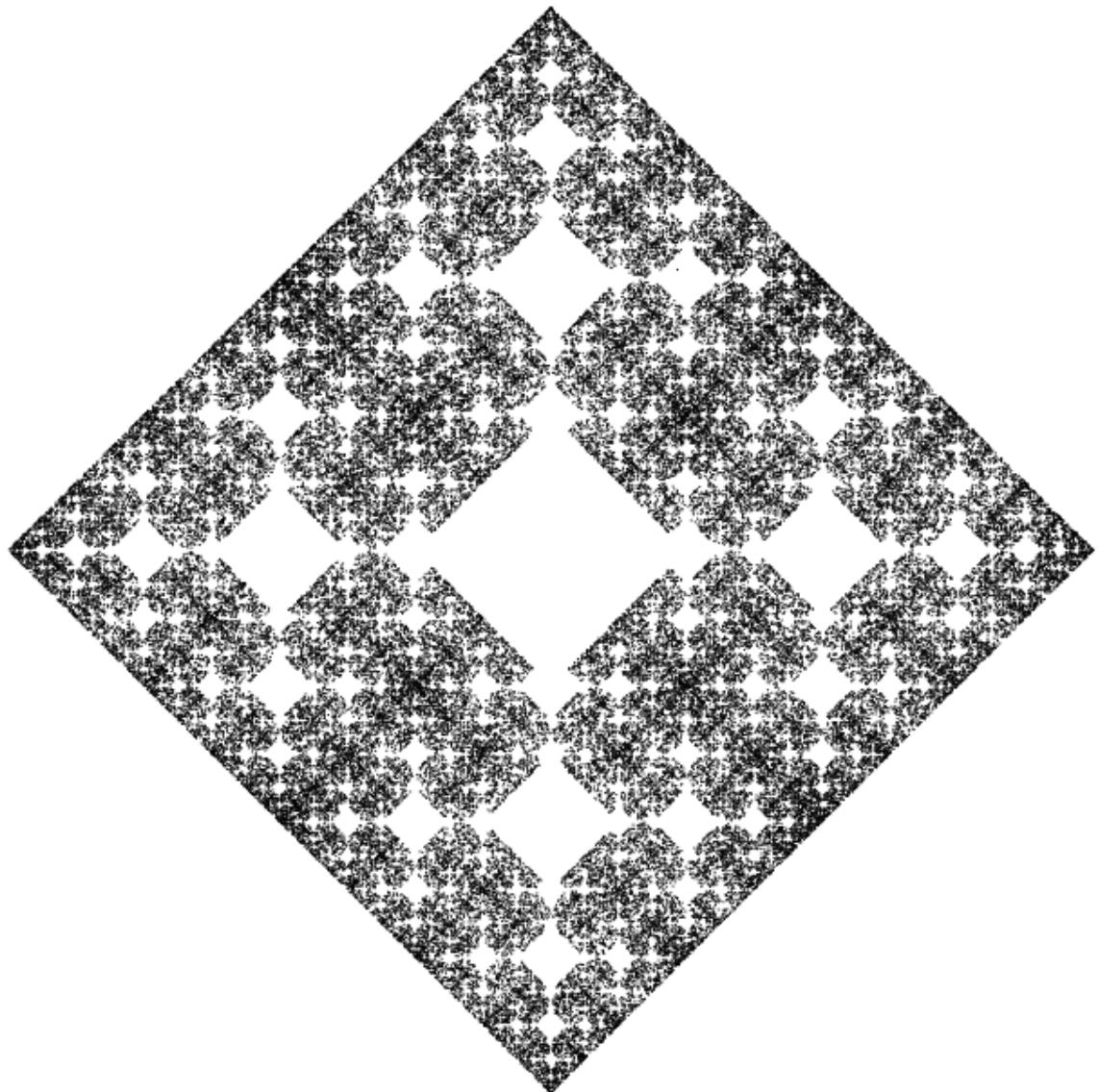
Jak widać, poza $q = (1)$ i $q = (3)$, gdzie jedno jest odbiciem lustrzanym drugiego, pozostałe fraktale są od siebie bardzo różne.

Teraz spróbujemy użyć tabeli ograniczeń nakładanych na przedostatnie i przedprzedostatnie udane losowanie.

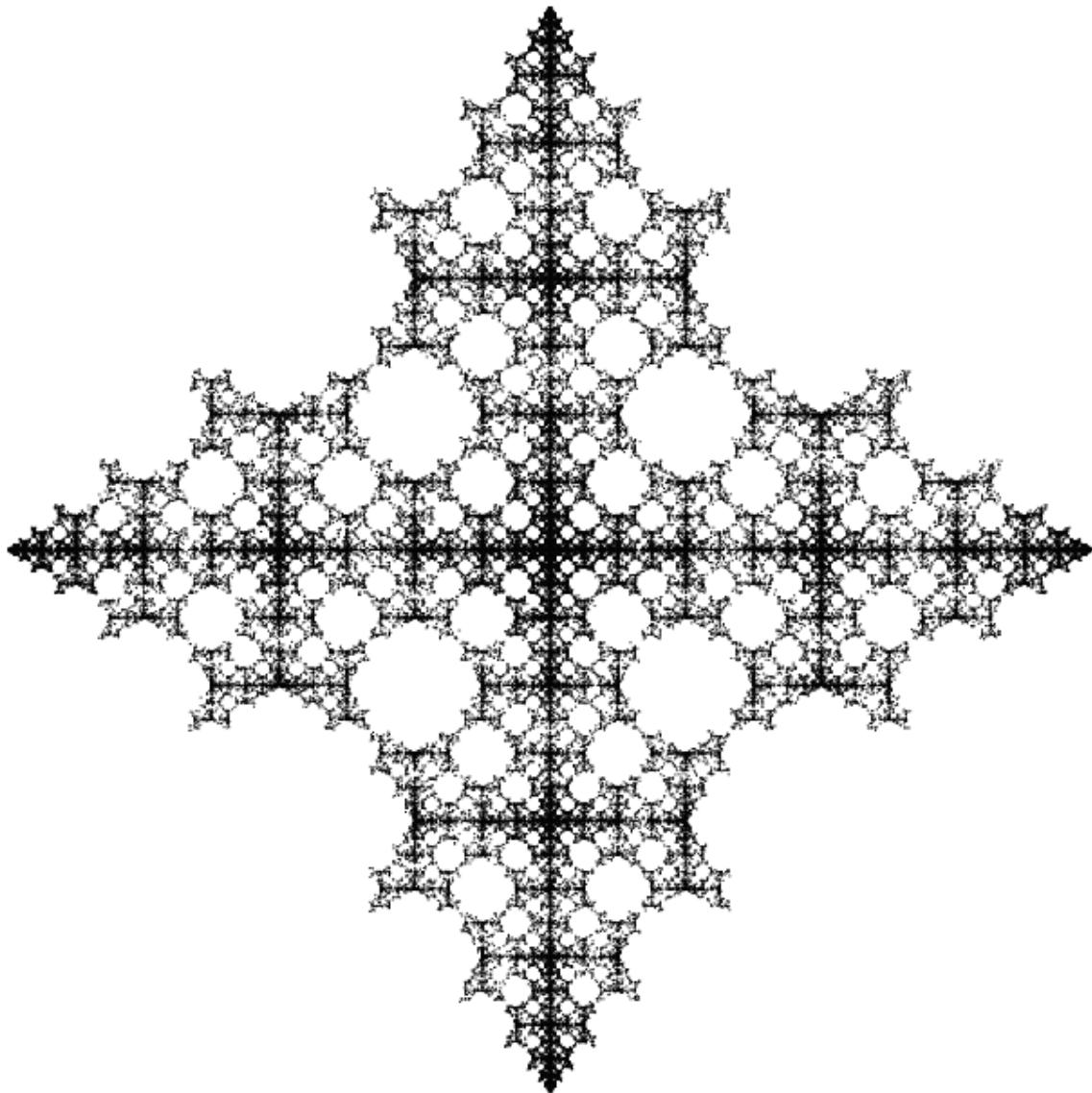
$$n = 4, r = 0.5, q = (1, 3)$$



$$n = 4, r = 0.5, q = (2, 2)$$



$$n = 4, r = 0.5, q = (3, 1)$$



Jak widać w obu przypadkach $q = (1, 3)$ i $q = (3, 1)$, o dziwo, otrzymaliśmy taki sam wynik bardzo różniący się od $q = (2, 2)$. Kolejna dziwna cecha chaosu, tak na parawdę nie wiadomo kiedy pojawi się jakiś niespodziewany porządek.

Bardziej złożone wielokąty

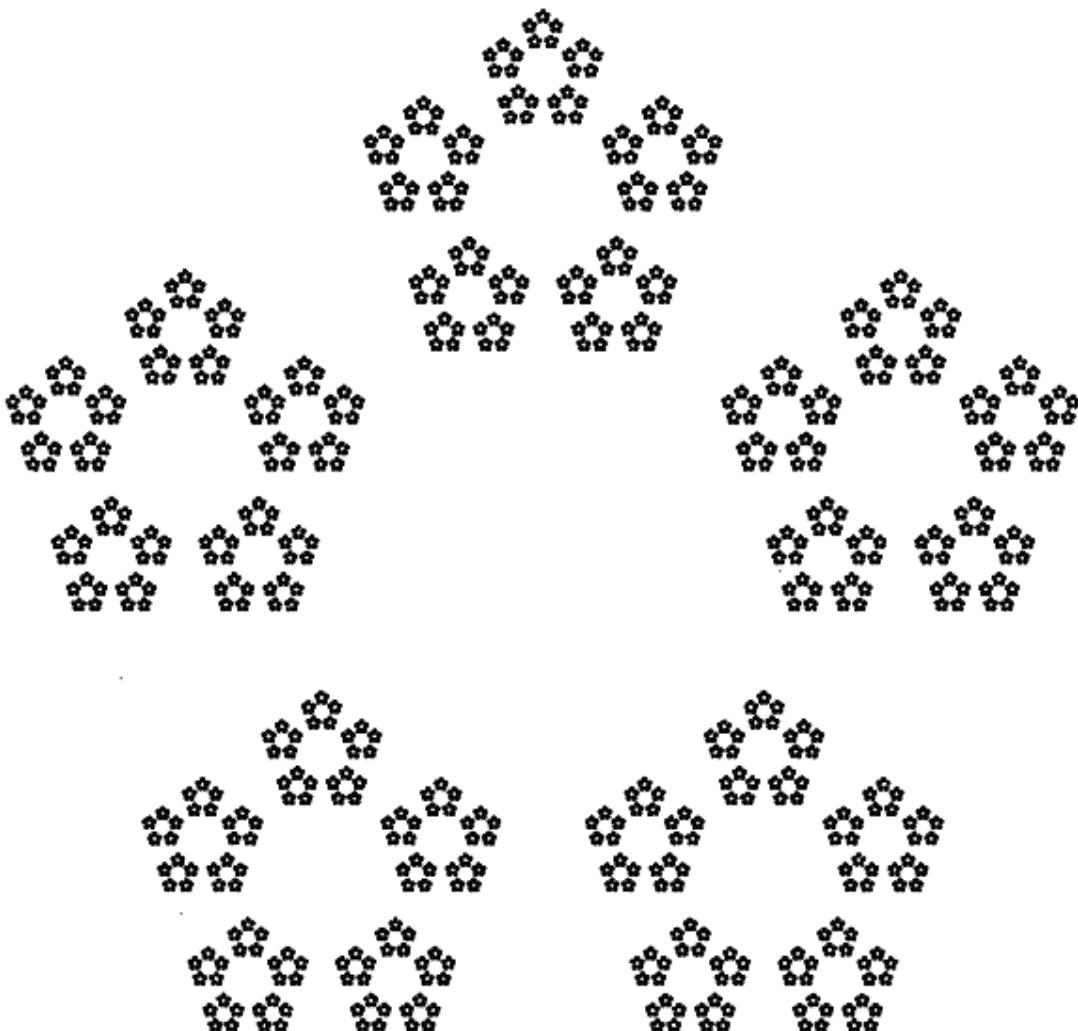
Pozwolę sobie zaprezentować tylko, kilka, bardziej złożonych wariantów dla pięciokąta foremnego, na początek z pustą tabelą ograniczeń.

W przypadku pięciokąta mam małą uwagę, mimo że na stronie *Wolfram*, proponuję użyć $r = 3/8$, lepszy rezultat (segmenty fraktala bardziej się ze sobą stykają) daje $r = \frac{1}{1+\varphi}$, gdzie φ oznacza słynną formułę *złotego podziału* (golden ratio)

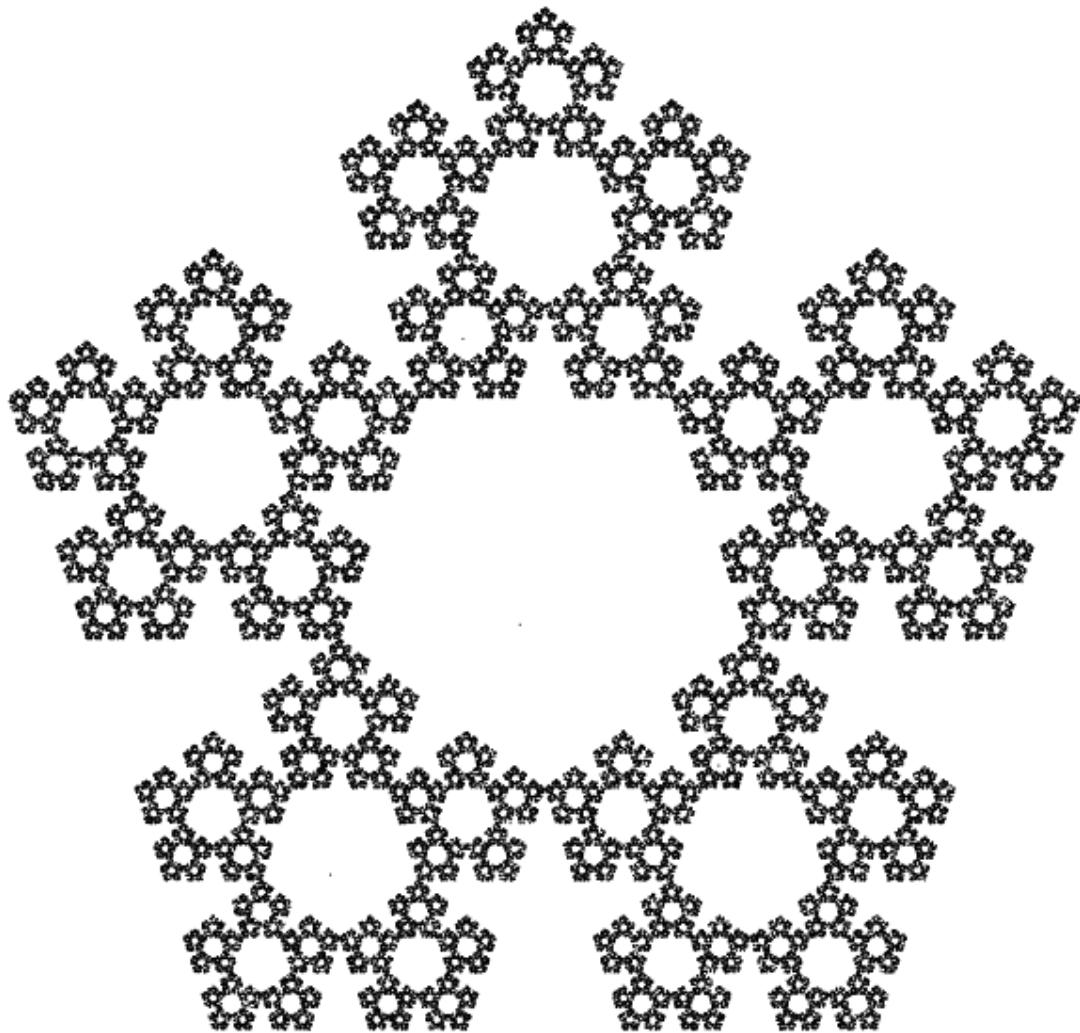
$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1,6180339887$$

Przykłady z pustą tabelą ograniczeń dla pięciokąta foremnego:

$$n = 5, r = \frac{1}{3}, q = ()$$

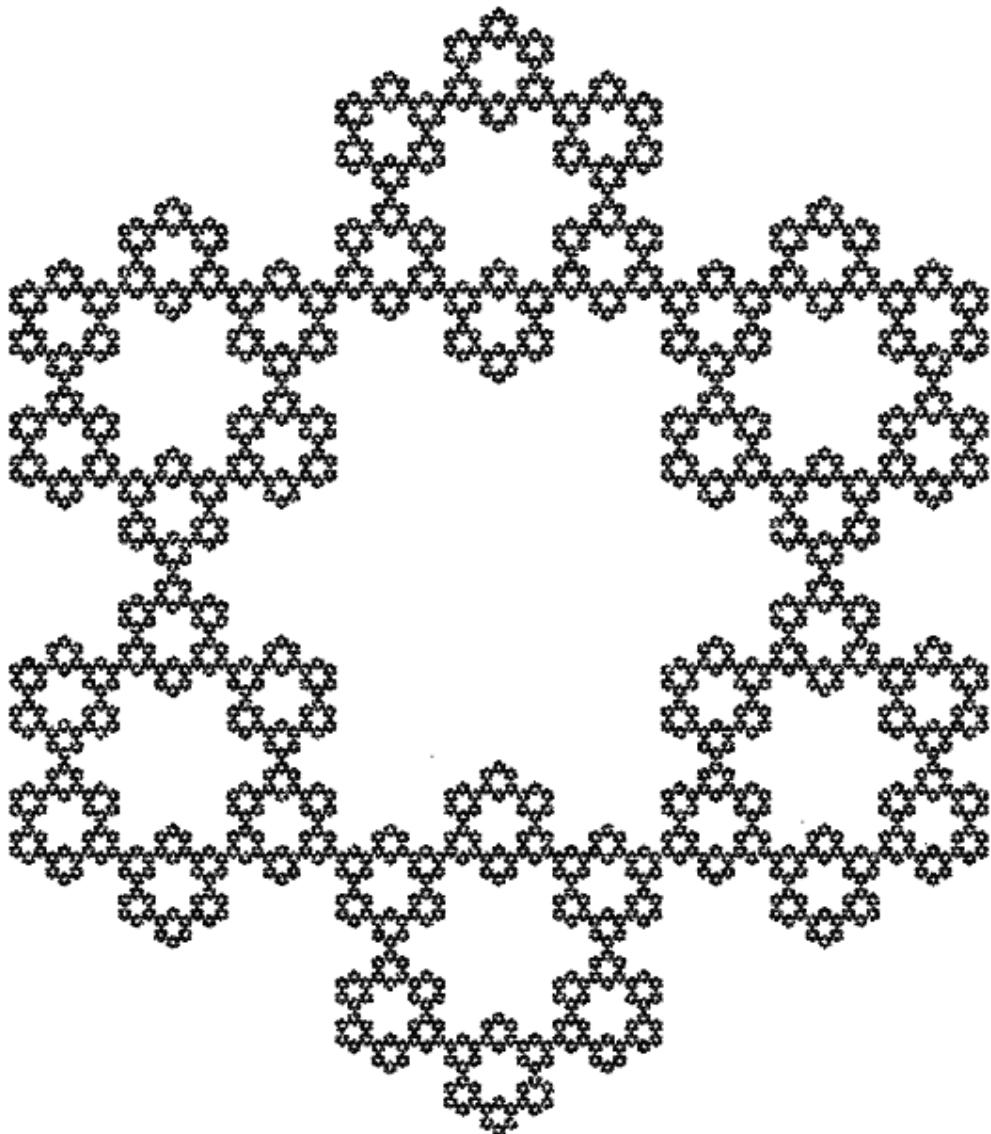


$$n = 5, r = \frac{1}{1 + \varphi}, q = ()$$



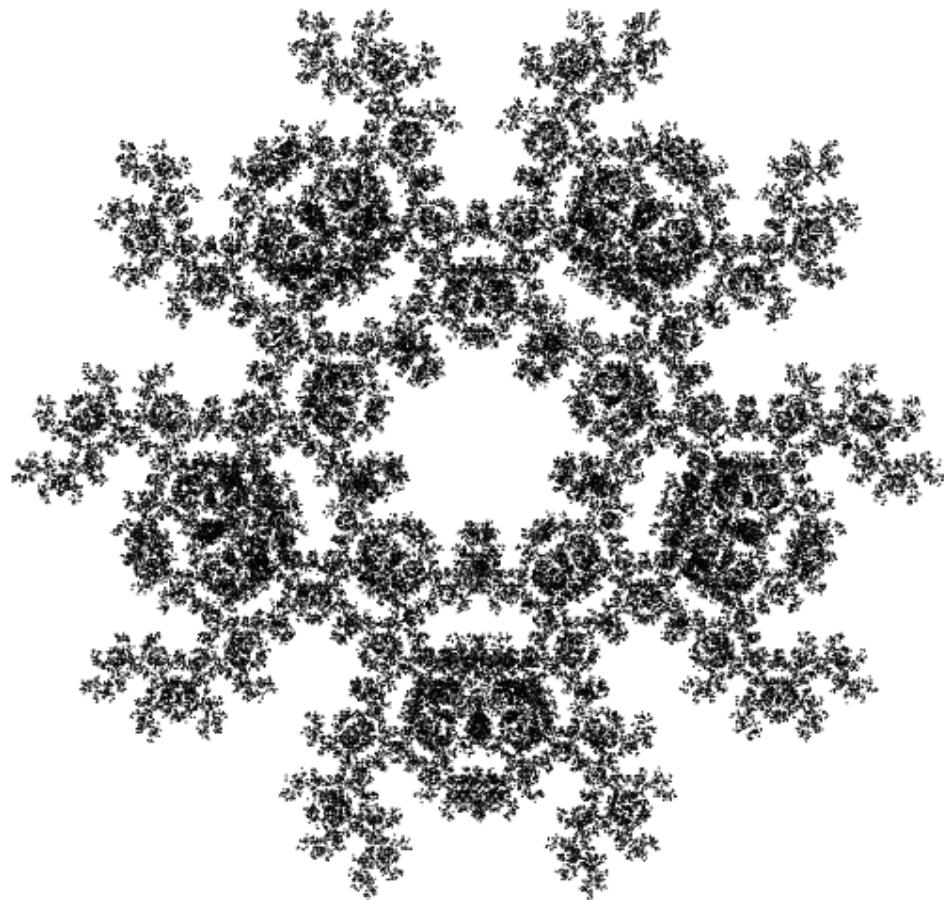
i sześciokąta foremnego:

$$n = 6, r = \frac{1}{3}, q = ()$$



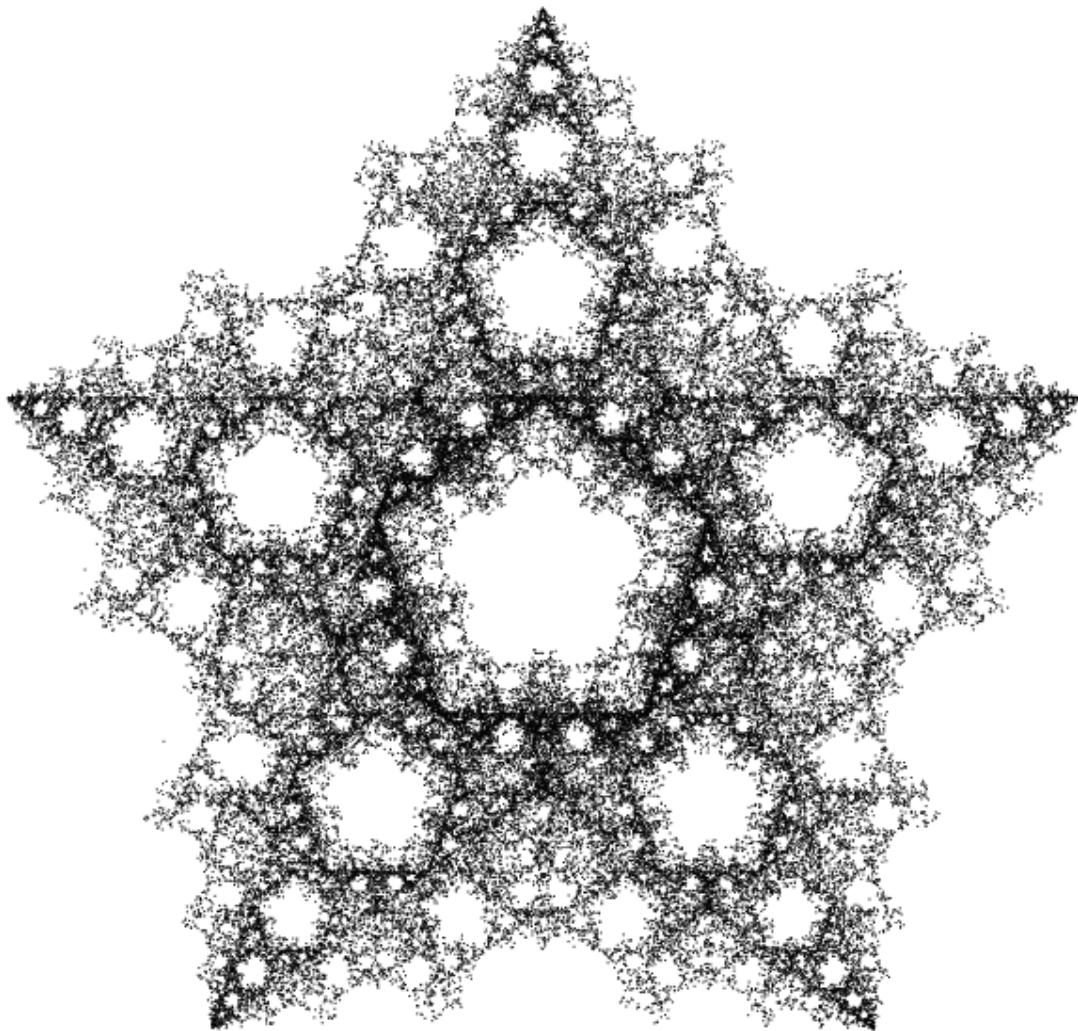
Przykłady z niepustą tabelą ograniczeń, jednoelementową:

$$n = 5, r = 0.5, q = (0)$$



i dwuelementową:

$$n = 6, r = 0.5, q = (1, 4)$$



Jak widać świat *gry w chaos* jest w stanie dostarczyć całkiem miłych wrażeń estetycznych, dać namiastkę torzenia dzieł plastycznych bez chodzenia na kółko plastyczne. Główne założenia, które urzekają swoją prostotą, pozwalają eksperymentować zarówno przez zmiany wartości parametrów n , r , q jak i generować trójwymiarowe fraktale w przestrzeni. Tu ograniczyłem się do najczęściej publikowanych obiektów.

Przed nami część praktyczna, czyli, jak takie wyrosłe z chaosu fraktale wygenerować na swoim ekranie.

Zrób to sam

Tytuł niniejszego rozdziału pewnie niewiele mówi młodszym czytelnikom ale dla wielu, którzy się on z popularnym niegdyś programem promującym majsterkowanie. Co było charakterystyczne w tym programie, otóż prowadzący, Pan Adam Słodowy pokazywał różne czynności na ekranie i wiele z nich faktycznie sam wykonywał, natomiast jeżeli dana czynność była długotrwała np. wycinanie dość skomplikowanych kształtów z puszki po konserwie, przeważnie

zaczynał tylko, po czym przerywał i mówił “ale ja już tutaj sobie przygotowałem” i wyciągał z pod stołu elegancko wykończony element.

Tak też i ja zrobię. Najpierw jednak o tym jak się do tego zabrałem.

Ponieważ algorytmy powinny być niezależne od technologii postanowiłem ograniczyć zestaw użytych narzędzi wg zasad:

- użyte technologie muszą być dostępne każdemu, za darmo na dowolnej platformie
- w obrębie użytych technologii nie będzie korzystane z dodatkowych bibliotek
- prezentacja ograniczy się do dwóch kolorów białego (tło) i czarnego (pisak)

Wybór padł na HTML5 i JavaScript. HTML5 bo ma dostępny element <canvas> wręcz idealny do tego typu operacji, javascript bo naturalnie się komponuje z HTML5 i może być użyty zarówno do opisania logiki problemu jak i do sterowania warstwą prezentacji.

Zgrubny podział na części:

- logiczną (logika biznesowa gry w chaos) skrypt `chaos-game.js`
- prezentacyjną (logika generowania fraktala i obsługi części wizualnej) skrypt `fractal.js`
- część spajająca (punkt wejścia ap likacji i kontener elementów graficznych) plik `index.html`

Logika gry w chaos

```
function ChaosGame(n, r, q) {  
  
    this.n = n;  
    this.r = r;  
    this.q = q;  
  
    this.vertices = [];  
    this.log = [0, 0];  
  
    this.init = function (width, height, radius) {  
        var angle = (2 * Math.PI) / this.n;  
  
        var center = {  
            x: width / 2,  
            y: height / 2  
        };  
  
        var vertex = {};  
  
        for (var i = 0; i < this.n; i++) {  
  
            if (i === 0) {  
                vertex = {  
                    x: center.x,
```

```

        y: center.y - radius
    );
}
else {
    vertex = {
        x: (vertex.x - center.x) * Math.cos(angle) - (vertex.y - center.y) * Math.sin(angle),
        y: (vertex.x - center.x) * Math.sin(angle) + (vertex.y - center.y) * Math.cos(angle)
    };
}

this.vertices.push(vertex);
};

this.getNextPoint = function (x, y) {

    var vertex = this.getVertex();

    return {
        x: vertex.x * (1 - this.r) + x * this.r,
        y: vertex.y * (1 - this.r) + y * this.r
    };
};

this.valid = function (v) {

    var enabled = false;

    if (this.q.length > 1 && q[0] !== q[1]) {

        enabled =
            this.log[0] !== ((v + (this.n + this.q[0])) % this.n) ||
            this.log[1] !== ((v + (this.n + this.q[1])) % this.n) ||
            this.log[0] !== ((v + (this.n + this.q[1])) % this.n) ||
            this.log[1] !== ((v + (this.n + this.q[0])) % this.n);

    }
    else {

        for (var i = 0; i < this.q.length; i++) {
            var c = this.log[i] !== ((v + (this.n + this.q[i])) % this.n);

            if (c === true) {
                return true;
            }
        }
    }
};
}

```

```

        }

        return enabled;
    };

    this.getVertex = function () {
        var v = 0;

        do {
            v = Math.floor(Math.random() * 10000) % this.n;
        } while (this.q.length > 0 && !this.valid(v));

        this.log.unshift(v);
        this.log.pop();

        return this.vertices[v];
    };
}

```

Generowanie fraktala

```

function Fractal(game, times, canvas, bgcolor, fgcolor) {
    this.game = game;
    this.canvas = canvas;
    this.times = times;
    this.fgcolor = fgcolor;
    this.bgcolor = bgcolor;

    this.render = function (x, y) {
        var context = this.canvas.getContext('2d');
        context.fillStyle = this.bgcolor;
        context.fillRect(0, 0, this.canvas.width, this.canvas.height);
        context.fillStyle = this.fgcolor;

        for (var i = 0; i < this.times; i++) {
            context.fillRect(x, y, 1, 1);

            var point = this.game.getNextPoint(x, y);

            x = point.x;
            y = point.y;
        }
    };
}

```

Entry point aplikacji

```
<!DOCTYPE html>
<html>
<head>
    <title>Chaos Game</title>
</head>

<body style="margin:auto;background:grey;text-align:center">

<canvas id="board"></canvas>

<script src="chaos-game.js"></script>
<script src="fractal.js"></script>

<script>

    /**
     * When n = 5, use golden ratio formula for better result
     * var PHI = (1 + Math.sqrt(5)) / 2;
     * var n = 3, r = 1 / (1 + PHI), q = [];
     */

    var n = 3, r = 0.5, q = [];

    var times = 100000;
    var size = 640;
    var radius = (size / 2) - 20;
    var bgcolor = '#FFF';
    var fgcolor = '#000';

    var canvas = document.getElementById('board');
    canvas.width = size;
    canvas.height = size;

    var game = new ChaosGame(n, r, q);
    game.init(canvas.width, canvas.height, radius);

    var fractal = new Fractal(game, times, canvas, bgcolor, fgcolor);
    fractal.render(0, 0);

</script>

</body>
</html>
```

Podsumowanie

1.1. Chaos game czyli nauka panowania nad chaosem, dla programistów, w 28 ramach spędzania wolnego czasu