13. Фракталы

Постановка задачи

Фракталы

Фракталами называют математические множества, обладающие свойством самоподобия: любая часть фрактала подобна всему фракталу целиком. Термин «фрактал» был введён Бенуа Мандельбротом в 1975 году и получил широкую известность с выходом в 1977 году его книги «Фрактальная геометрия природы» [1]. Однако многие фракталы были известны и ранее.

В этом задании требуется разработать программу, которая строит изображения нескольких фракталов. Будут использованы фракталы, построение которых естественным образом осуществляется с помощью рекурсивных процедур.

Первым фракталом будет фрактальное дерево, которое иногда также называют Пифагоровым деревом. Первые итерации построения этого фрактала показаны на рисунке 1.

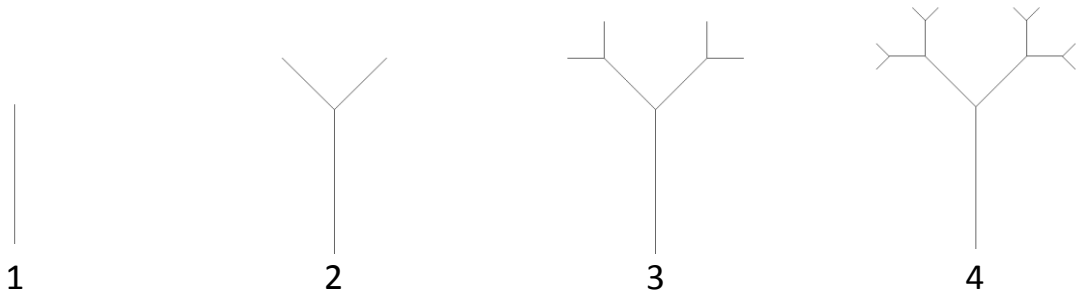


Рис. 1. Первые итерации построения фрактального дерева

Процесс начинается с рисования вертикальной линии определённой длины. Затем к верхнему концу этой линии добавляют две линии меньшей длины, направленные под углом 45° направо и налево от направления исходной линии. И далее процесс продолжается – на концах каждой из линий добавляют ещё две линии и так далее. Теоретически, такой процесс построения фрактала должен продолжаться бесконечно. На практике его обычно ограничиваются конечным числом этапов.

Фрактальное дерево является примером класса фракталов, которые называют L-системами, или системами Линденмайера, по фамилии открывшего их в 1968 году венгерского ботаника. Линденмайер разработал

теорию L-систем в ходе изучения роста и развития растений [2]. Многие из фракталов этого класса похожи на реальные растения, их часто применяют для создания изображений растений (рис. 2).

Рис. 2. Пример изображения растения, сгенерированного с помощью L-системы

Другим классическим примером фрактала из класса L-систем является кривая Коха (рис. 3).

Рис. 3. Первые итерации построения кривой Коха

Для построения этой кривой следует взять отрезок и заменить его центральную треть ломаной, повторяющей форму равностороннего треугольника. Затем, каждый из получившихся отрезков подвергается такому же преобразованию.

Если начать построение не с отрезка, а с равностороннего треугольника, то получится фигура, известная как снежинка Коха (рис. 4).

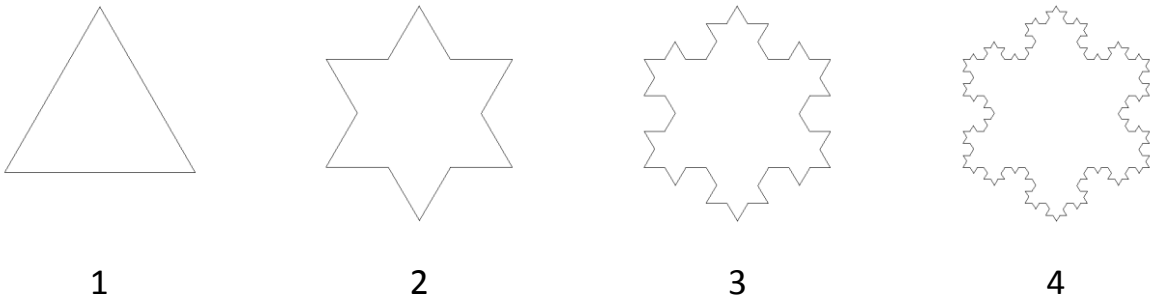


Рис. 4. Первые итерации построения снежинки Коха

Последний фрактал не будет относиться к классу L-систем. Этот фрактал известен как ковёр Серпинского (рис. 5).

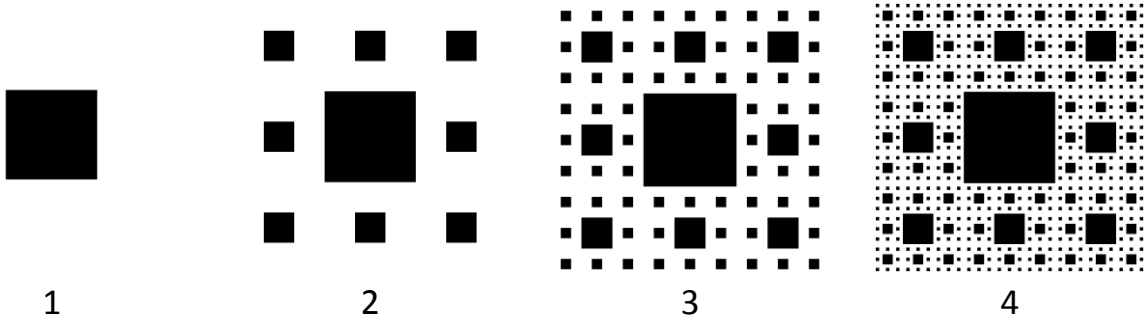


Рис. 5. Первые итерации построения ковра Серпинского

Построение Ковра Серпинского начинается с квадрата. На первом шаге квадрат разбивается сеткой 3×3 ячейки и центральная ячейка удаляется (или закрашивается). Затем процесс рекурсивно повторяют для 8 оставшихся ячеек.

Ковёр Серпинского, так же как и фракталы из класса L-систем, используют для создания компактных, но эффективных антенн (рис. 6).

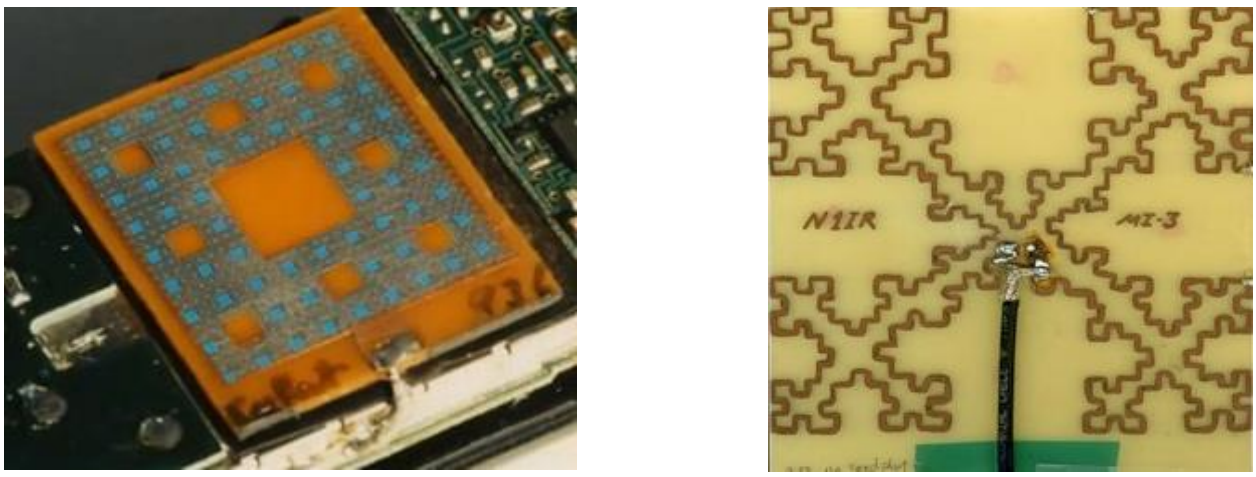


Рис. 6. Фрактальные антенны

Архитектура программы

Обратите внимание, что алгоритмы построения этих фракталов рекурсивны: они состоят из отдельных итераций, на каждой итерации могут быть нарисованы какие-то «обычные» компоненты (линии, квадраты, …), а затем осуществляются рекурсивные вызовы для построения оставшихся частей фигуры. При рекурсивном вызове размер фигуры уменьшается, могут измениться её положение и ориентация.

В теории процедура построения фрактала бесконечна. На практике бесконечное вычисление невозможно, поэтому ограничиваются несколькими шагами вглубь рекурсии. На последнем этапе вместо рекурсивного вызова выполняется рисование базовой фигуры, в рассматриваемых фракталах это те же самые «обычные» компоненты, которые используются на предыдущих шагах рекурсии (можно также использовать объект, получаемый на втором шаге рекурсии).

Сама структура рекурсивного алгоритма рисования фракталов одинакова для всех. Различными являются:

- расположение и ориентация начального объекта;
- базовая фигура;
- детали рекурсивного шага: где нарисовать базовые компоненты, а где рекурсивно продолжить построение.

Конечно, можно нарисовать каждый фрактал с помощью одной рекурсивной функции. Но при этом придётся каждый раз повторить те действия, которые являются общими для всех. Чтобы избежать такого повторения в данном случае можно использовать классы и наследование.

Наследование

При разработке с использованием классов случается так, что разные классы содержат одинаковые поля данных и/или методы. Например, в учётной системе отдела кадров университета могут быть записи про студентов и преподавателей, функциональность которых может в значительной степени пересекаться:

	Преподаватель	Студент
Поля данных	**Фамилия, имя, отчество** **Паспортные данные** **Адрес** Должность Дисциплины	**Фамилия, имя, отчество** **Паспортные данные** **Адрес** Курс Группа Рейтинг
Методы	**Изменить адрес** Закрепить за дисциплиной	**Изменить адрес** Изменить рейтинг

В данном случае наличие повторяющихся полей данных и методов вызвано тем, что преподавателей и студентов можно естественным образом объединить более общим понятием – люди. Можно сказать, что понятия «Студент» и «Преподаватель» уточняют понятие «Человек».

В объектно-ориентированных языках программирования для выражения такого отношения между объектами используется наследование. При этом создаются несколько классов, соответствующих как общим понятиям («человек») так и более конкретным («преподаватель» и «студент»).

Класс, соответствующий общему понятию, называют предком (англ. parent), базовым классом (англ. base class) или суперклассом (англ. parent class). В этот класс помещают общие свойства и методы.

Классы, соответствующие уточнённым понятиям, называют производным классом (англ. derived class), классом потомком (англ. child class) или подклассом (англ. *subclass*). В таких классах помещают методы и свойства, характерные только для этого конкретного типа. При этом механизм наследования автоматически добавляет в подклассы методы и поля базового класса.

Отношение наследования удобно представлять в виде графических диаграмм, подобных показанной на рисунке 7. В таких диаграммах класс показывается прямоугольником, разделённым на 3 секции. В верхней секции указывается имя класса, в средней – его поля данных, а в нижней – методы. Отношение наследования показывается с помощью линии с треугольником, вершина которого указывает на суперкласс.

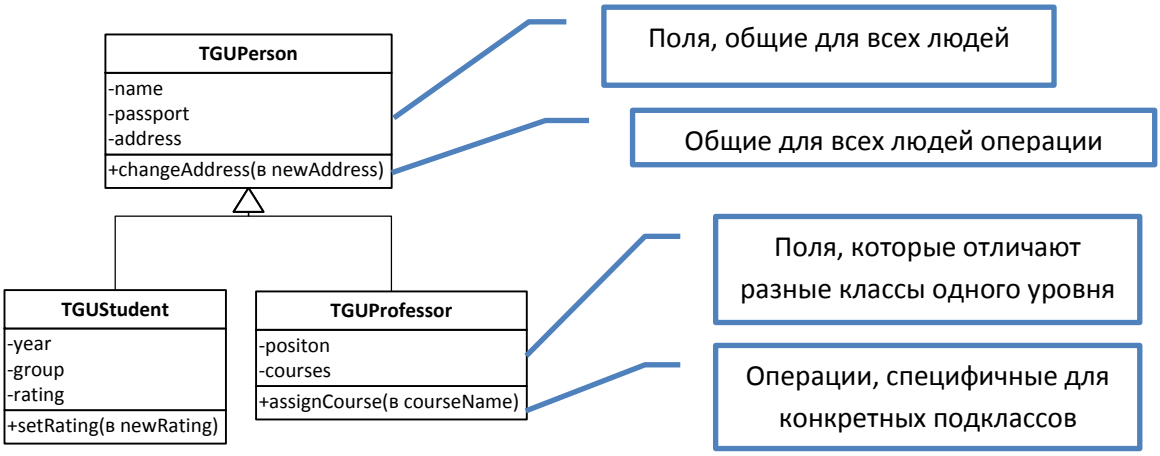


Рис. 7. Пример диаграммы наследования классов

Чтобы реализовать наследование между классами на языке программирования Python необходимо сделать следующее:

1) напишите реализацию суперкласса – это простой класс без каких-либо особенностей;
2) реализуйте производные классы, при этом:
    a) после имени класса укажите в круглых скобках имя родительского класса;
    b) в конструкторе производного класса необходимо вызвать конструктор родительского класса с помощью специального синтаксиса, как показано в примере ниже;
    c) в методах производных классов можно использовать поля данных и методы (кроме конструктора) родительского класса так, как будто бы они являются методами этого класса.

Далее показан пример реализации наследования классов на языке программирования Python.

```
# Суперкласс – «человек».
class TGUProfessor:
    def __init__(self, name, passport, address):
        self.name = name
        self.passport = passport
        self.address = address

    def changeAddress(self, newAddress):
        self.address = newAddress
```

```
# Производный класс – «преподаватель».
class TGUStudent(TGUPerson):    # Указываем базовый класс!
    def __init__(self, name, passport, address, position):
        # Вызов конструктора базового класса.
        super().__init__(name, passport, address)
        # Обычная работа конструктора
        self.position = position
        self.courses = []

    def assignCourse(courseName):
        self.courses.append(courseName)
```

Наследование и рисование фракталов

Для того, чтобы выделить базовую структуру алгоритма построения фракталов следует разбить процедуру рисования на небольшие действия и те из них, которые окажутся одинаковыми, поместить в суперкласс. Различающиеся действия помещают в производные классы, создав по одному производному классу для каждого фрактала.

На рисунке 8 показана иерархия классов, которые будут использованы в этом задании. Общие для всех фракталов методы и данные вынесены в суперкласс `Fractal`. От этого класса унаследованы классы `CantorSet` (рисует множество Кантора, которое будет рассмотрено в разделе «предоставляемый код»), `FractalTree` (фрактальное дерево), `KochLine` (кривая Коха) и `SerpinskiCarpet` (ковёр Серпинского). Класс `KochSnowflake` (снежинка Коха) наследуется от класса `KochLine`. Рассмотрим эти классы подробнее.

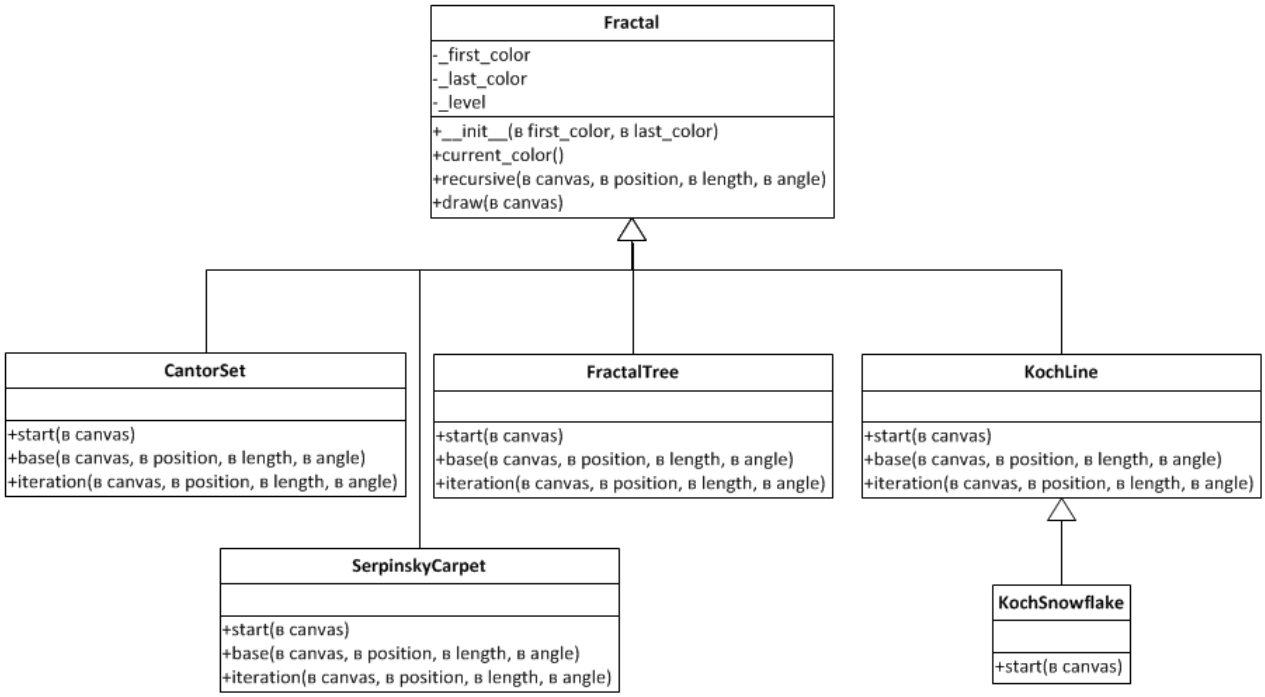


Рис. 8. Иерархия классов программы построения фракталов

Так как основной задачей всех классов является рисование фракталов, им не придётся хранить много данных. В данном случае все переменные класса собраны в базовом классе `Fractal`. Переменная `_level` будет хранить текущий уровень рекурсии. Максимальный уровень, до которого будет производиться спуск, будет храниться в глобальной переменной `max_levels`.

Для улучшения эстетических свойств фракталов будем рисовать их с использованием разных цветов для каждого уровня. Переменные `_first_color` и `_last_color` будут хранить цвета для первой и последней итерации, цвета для промежуточных итераций будут вычисляться. Для упрощения вычислений цвет будет храниться в виде кортежа `(R,G,B)`, где R, G и B – значения яркости красной, зелёной и синей компонент.

Конструкторы всех классов будут получать значения цветов для первой и последней итерации. Не забывайте вызывать конструктор суперкласса из конструкторов подклассов и передавать ему эти параметры.

Метод `draw(canvas)` класса Fractal будет начинать процесс рисования. Он будет вызываться из обработчика рисования холста и получит в качестве аргумента холст. Задав начальное значение переменной _level, он вызовет метод `start(canvas)`, который должен быть определён в подклассе и который должен начать сам процесс рисования.

Метод `recursive(canvas, position, length, angle)` отвечает за рисование рекурсивной части фрактала. Он должен отслеживать изменение уровня рекурсии в переменной _level и в зависимости от него вызывать или метод `iteration()`, или метод `base()` из подклассов. Метод `base()` будет вызываться на последнем шаге и должен нарисовать базовый элемент фрактала. Метод `iteration()` отвечает за реализацию промежуточных итераций: он должен отрисовать фиксированную часть (если она присутствует) и вызвать `recursive()` для рисования «рекурсивных» частей.

Таким образом, в процессе рисования фрактала будет вначале вызван метод `draw()`, затем `start()`, затем `recursive()`, `iteration()`, снова `recursive()`, `iteration()` и так далее, пока не дойдёт до последнего уровня, на котором `recursive()` вызовет `base()`. На рисунке 9 показано, какие вызовы будут выполняться для построения трех уровней фрактального дерева.

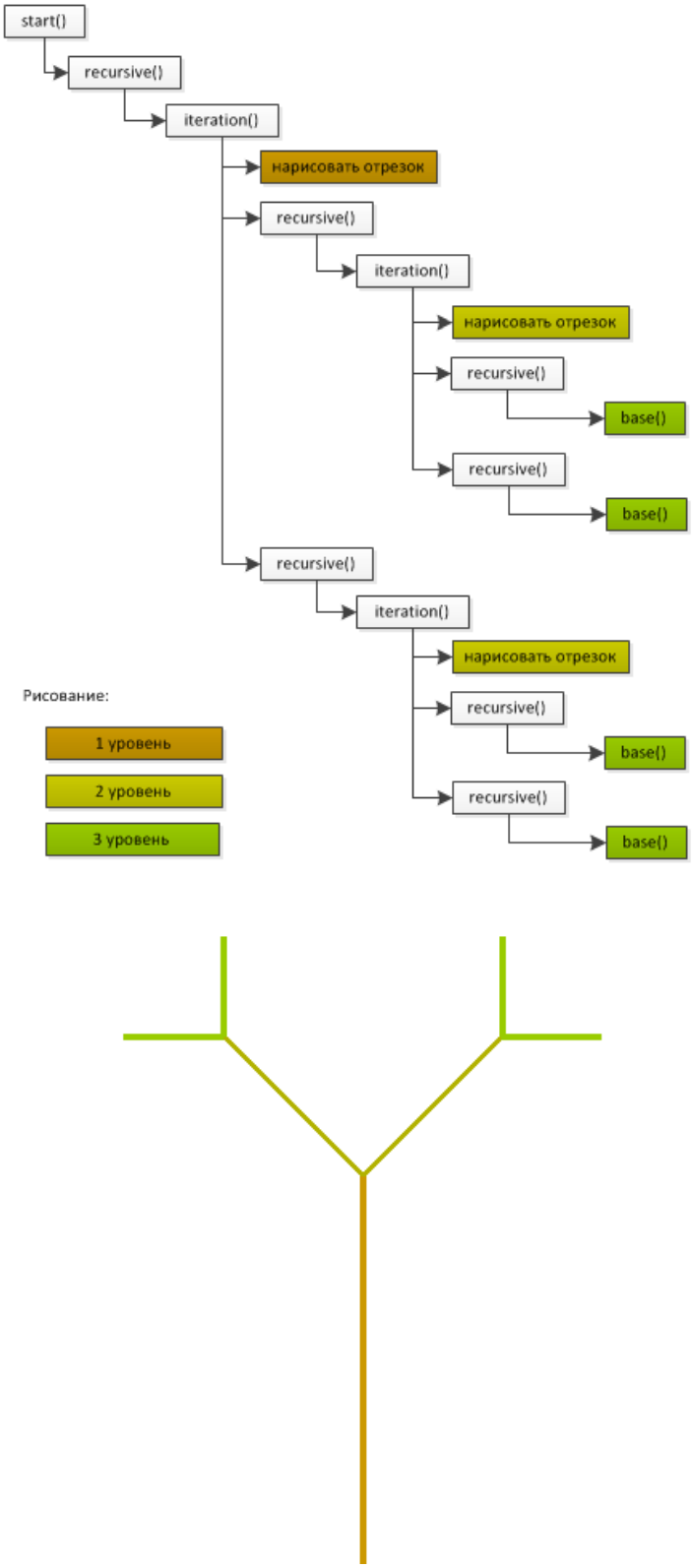


Рис. 9. Вызовы методов при построении фрактального дерева до 3-го уровня

Методы `recursive()`, `iteration()` и `base()` получают один и тот же набор аргументов. Первым из них является `canvas` – холст, на котором производится рисование. Оставшиеся три параметра описывают размер и расположение той части фрактала, которую необходимо нарисовать:

`position` – начальная точка (кортеж из двух значений `(x, y)`); `length` – длина части и `angle` – угол (направление) в радианах. Этих трех параметров достаточно, чтобы задать положение для всех используемых в задании фракталов (для ковра Серпинского угол не потребуется).

Обратите внимание, что класс `KochSnowFlake` наследуется от класса `KochLine` и переопределяет только один метод – `start()`. Так сделано потому, что снежинка Коха отличается от кривой Коха только начальной фигурой, процедура построения у них одинаковая. Поэтому, сделав класс `KochLine`, можно использовать все его возможности для построения снежинки Коха, заменив только один метод.

Предоставляемый код

Для разработки кода следует использовать шаблон fractal_painter_template.py.

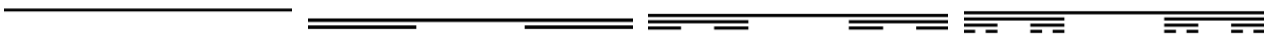
Шаблон содержит заголовки классов Fractal, `FractalTree`, `KochLine`, `KochSnowflake` и `SerpinskiCarpet`.

В шаблоне содержится полная реализация класса CantorSet, которую можно использовать для проверки правильности реализации класса `Fractal`.

В конце шаблона находится реализация графического интерфейса программы.

Класс CantorSet

Класс CantorSet реализует один из вариантов фрактала, соответствующего множеству Кантора [3]. Процесс построения этого фрактала показан на рисунке 10.



1 2 3 4

Рис. 10. Первые итерации построения множества Кантора

Построение фрактала начинается с горизонтального отрезка. Длина отрезка разбивается на три равные части, под первой и последней из них рисуется следующий уровень фрактала.

Рассмотрим реализацию класса, отображающего этот фрактал.

Конструктор

```
class CantorSet(Fractal):
    def __init__(self, first_color, last_color):
        Fractal.__init__(self, first_color, last_color)
```

Конструктор класса просто передаёт параметры конструктору суперкласса.

Метод start()

```
class CantorSet(Fractal):
    def start(self, canvas):
        self.recursive(canvas, (0, SIZE / 3), SIZE, 0)
```

Метод `start()` отвечает за начало рисования. При выводе этого фрактала позиция (`position`) будет рассматриваться как левая точка отрезка очередного уровня; размер (`length`) – длина этой прямой; угол (`angle`) не будет использован.

Таким образом, данный метод начинает построение фрактала, вызвав рекурсивную процедуру для рисования фрактала, начиная с отрезка шириной с холст (переменная `SIZE` в шаблоне), расположенного на 1/3 высоты холста.

Метод base()

```
class CantorSet(Fractal):
    def base(self, canvas, position, length, angle):
        canvas.draw_line(position, (position[0] + length,
position[1]),
                         5, self.current_color())
```

Этот метод отвечает за рисование базового элемента фрактала. В данном случае это просто отрезок прямой. Отрезок начинается с точки `position` и имеет длину `length`.

Для выбора цвета прямой используется метод `current_color()`, реализованный в классе `Fractal`.

Метод iteration()

```
class CantorSet(Fractal):
    def iteration(self, canvas, position, length, angle):
        self.base(canvas, position, length, angle)
        self.recursive(canvas, (position[0], position[1] + 10),
                       length / 3, 0)
        self.recursive(canvas, (position[0] + 2 * length / 3,
```

```
                    position[1] + 10),
            length / 3, 0)
```

Самый сложный метод этого класса включает три строки и отвечает за реализацию одной итерации построения фрактала. Его работу можно понять, обратившись к рисунку 11.

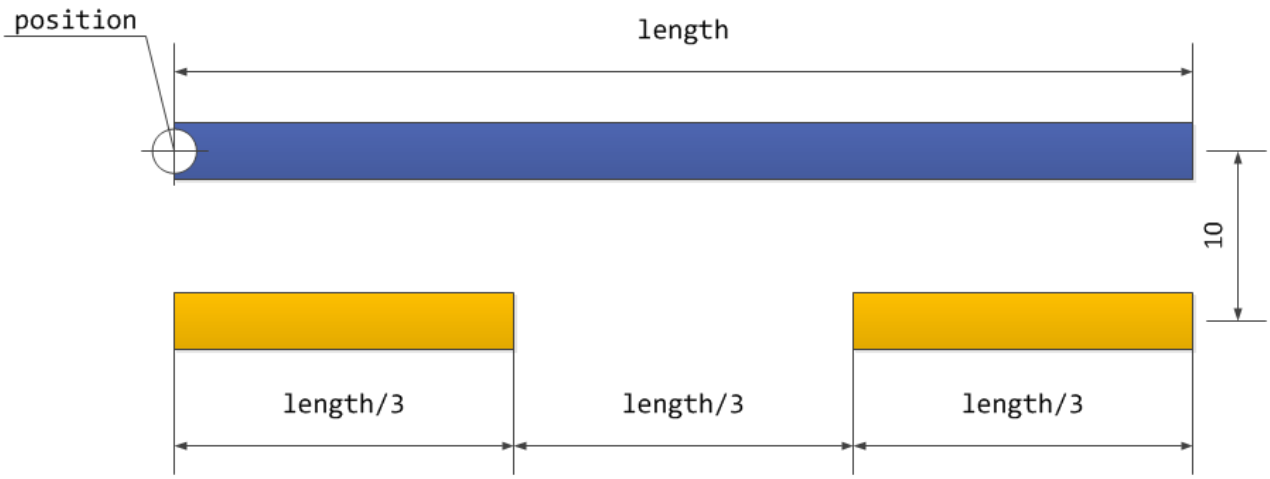


Рис. 11. Итерация построения фрактала «множество Кантора»

Итерация начинается с рисования отрезка из точки `position` длиной `length`. Этот отрезок показан синим цветом на рисунке 10. Для вывода отрезка используется метод `base()`, так как он строит в точности такой отрезок.

Далее необходимо рекурсивно провести процедуру построения фрактала в местах, показанных (рис. 9) жёлтым цветом. Для каждого из них вызывается метод `recursive()` класса `Fractal`. При этом:

- длина отрезка уменьшается в три раза;
- координата по высоте увеличивается (опускается на экране) на 10 пикселей;
- координата по ширине рассчитывается исходя из положения исходного отрезка и размеров частей.

Интерфейс программы

В конце шаблона находится код реализации интерфейса пользователя. Интерфейс включает кнопки для выбора типа фрактала и поле ввода для установки максимального уровня рекурсии. Уровень рекурсии хранится в глобальной переменной `max_levels` и по умолчанию равен 3.

Фрактал для рисования находится в глобальной переменной `f`. Объект, на который указывает эта переменная, пересоздаётся обработчиками кнопок:

- cantor_set() – множество Кантора;
- fractal_tree() – фрактальное дерево;
- koch_line() – кривая Коха;
- koch_snowflake() – снежинка Коха;
- serpinski_carpet() – ковёр Серпинского.

При создании фрактала в каждой функции задаётся своя цветовая схема.

Обратите внимание, что перед созданием окна вызывается функция cantor_set(), соответственно при запуске программа будет отображать множество Кантора.

Обработчик рисования `draw()` вызывает метод `draw()` текущего фрактала: объекта `f`.

Рекомендованный порядок выполнения задания

Работа над проектом разбита на три этапа.

Этап 1. Класс Fractal

На этом этапе предстоит реализовать основные методы класса `Fractal`.

Шаблон содержит реализацию метода `__init__` для класса `Fractal`. Конструктор принимает на вход два обязательных аргумента: цвета для компонентов первого и последнего уровней.

1. Реализуйте метод draw(canvas), который рисует фрактал.
   В этом методе следует установить уровень рекурсии (переменная `self._level`) в `0` и вызвать метод self.start(), который будет определён в подклассах.
2. Реализуйте метод recursive(canvas, position, length, angle).
   В первую очередь этот метод должен увеличить на единицу текущий уровень рекурсии.
   Затем проверьте, достиг ли уровень максимального значения, заданного глобальной переменной `max_levels`.
   Если нет, следует вызвать метод `self.iteration()` для выполнения

очередной итерации. Если максимальная глубина уже достигнута, следует вызвать метод `self.base()`.

Перед выходом из `recursive()` уменьшите уровень итерации на 1.

Если оба метода реализованы верно то при запуске программы будет нарисовано множество Кантора, как показано на рисунке 9.

При ошибках вычисления текущего уровня или момента остановки рекурсии программа может зависнуть или нарисовать множество не целиком. Перед продолжением работы убедитесь, что всё работает правильно.

Если множество Кантора в чёрно-белом варианте рисуется правильно, можно добавить к программе цвет. Для этого необходимо реализовать метод `current_color()`. Он должен рассчитать цвет исходя из:

- текущего уровня рекурсии, заданного переменной self._level;
- цвета для первого уровня, заданного переменной self._first_color;
- цвета для последнего уровня, заданного переменной self._last_color.

Цвета для промежуточных уровней будем вычислять по линейному закону, изменяя яркость каждой компоненты от начального значения к конечному пропорционально глубине спуска.

Для этого, вначале вычислите значение alpha так, чтобы оно изменялось от `0` при `self._level==1`, до `1` при self._level==max_level.

Используя alpha, с помощью вспомогательной функции `gradient()` вычислите значения компонент цвета. Помните, что в программе цвет хранится в виде кортежей, первая компонента которых соответствует яркости красного, вторая – зелёного, а третья – синего цвета.

Библиотека SimpleGui поддерживает текстовое представление цвета. Поэтому преобразуйте полученное значение в строку вида «`rgb(R,G,B)`», где `R`, `G` и `B` - вычисленные значения яркости компонент, и верните эту строку.

Если всё было проделано корректно, при запуске программы множество Кантора будет отображено, как показано на рисунке 12.

Рис. 12. Множество Кантора, окрашенное по уровням рекурсии

Убедитесь, что отрезок верхнего уровня окрашен чистым красным цветом (`rgb(255, 0, 0)`), а отрезки нижнего – чистым зелёным (`rgb(0, 255, 0)`). Если это не так, в первую очередь проверьте правильность вычисления alpha.

Проверьте, что программа работает при числе уровней, равном 1. В этом случае для отображения можно выбрать любой из цветов.

Этап 2. Фрактальное дерево

Завершив работы с суперклассом Fractal, можно переходить к реализации его подклассов, рисующих разные фракталы. Начать лучше с самого простого в реализации – фрактального дерева.

При построении этого фрактала ориентируйтесь на схему на рисунке 13.

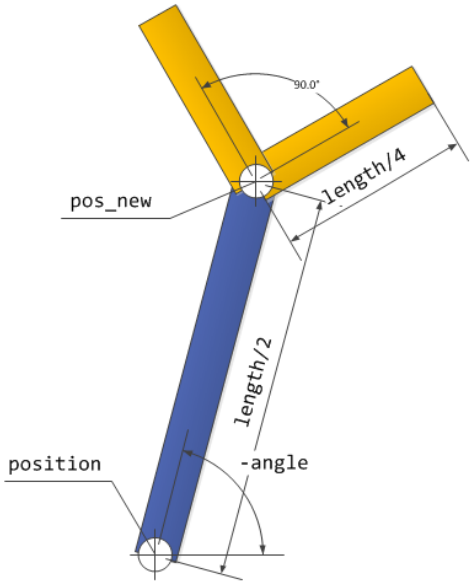


Рис. 13. Итерация построения фрактального дерева

В отличие от множества Кантора при построении фрактального дерева ориентация его компонентов будет меняться, поэтому в расчётах будет использован угол, задаваемый аргументом `angle`. Так как в системе координат холста ось ординат направлена вниз, то и направления углов оказываются «перевёрнутыми»: положительное значение угла означает движение по часовой стрелке, а отрицательное – против часовой. Это будет

существенно при определении ориентации первого компонента фрактала. Далее направления будут вычисляться относительно предыдущего.

Также необходимо помнить, что углы в программе задаются в радианах, 180°=π радиан.

Реализуйте класс FractalTree следующим образом:

1. Реализуйте конструктор класса. Необходимо передать параметры конструктору суперкласса.

2. Реализуйте метод `start()`, который запускает рисование фрактала.
   Рекомендуется расположить начальную точку на середине нижней стороны холста, в качестве размера взять не менее половины размера холста и ориентировать первый сегмент вверх.

3. Реализуйте метод `base()`, который рисует базовую фигуру.
   Этот метод должен нарисовать отрезок, показанный синим цветом на рисунке 13. Его начальной точкой является точка `position`.
   Чтобы рассчитать конечную точку (`pos_new`), воспользуйтесь вспомогательной функцией `angle_to_vector(angle, length)`, которая возвращает вектор, ориентированный по направлению `angle` и имеющий длину `length`. Добавьте этот вектор к координатам точки `position`, чтобы получить координаты точки `pos_new`.
   При рисовании линии не забудьте использовать `self.current_color()` для задания цвета.

4. Реализуйте метод `iteration()`, который выполняет одну итерацию. Следуйте схеме, представленной на рисунке 11.
   Для рисования синего отрезка используйте метод `base()`.
   Далее два раза вызовите метод `recursive()`, чтобы построить рекурсивные части, обозначенные жёлтым цветом. При этом необходимо скорректировать все параметры расположения частей фрактала:
   - новой позицией будет точка `pos_new`;
   - размер должен уменьшится в два раза;
   - углы для фрагментов вычисляются добавлением и вычитанием 45° (не забудьте перевести в радианы) из текущего угла.

Координаты точки `pos_new` уже вычисляются в методе `base()`, чтобы не вычислять их второй раз, можно вернуть их из метода `base()` и использовать в `iteration()`.

Если всё было выполнено верно, то после запуска программы и нажатия на кнопку «Фрактальное дерево» можно увидеть этот фрактал.

Поэкспериментируйте с пропорциями и ориентацией компонентов фрактала. Ветви дерева не обязательно должны расходиться под одинаковым углом. Можно добавить к дереву и новые ветви. Такие вариации могут порождать различные красивые структуры.

Этап 3. Кривая Коха

Теперь обратимся к следующему фракталу – кривой Коха. Структура итерации для него показана на рисунке 14.

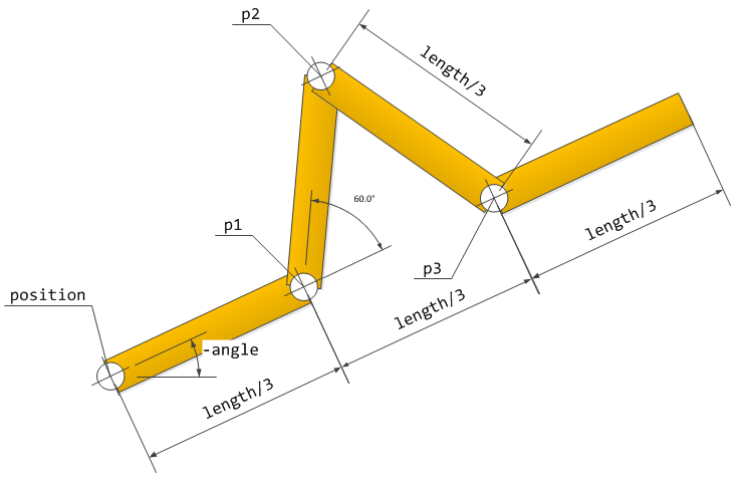


Рис. 14. Структура итерации при построении кривой Коха

1. Реализуйте конструктор класса. Необходимо передать параметры конструктору суперкласса.

2. Реализуйте метод `start()`, который запускает рисование фрактала. Расположите начальную точку на левой стороне холста в его нижней части. В качестве начального размера используйте полную ширину холста. Начальный угол равен 0 – направление вдоль оси OX.

3. Реализуйте метод `base()`, который рисует базовую фигуру. Хотя итерация кривой Коха не содержит никаких «обычных» линий, не забывайте, что основная задача метода `base()` – нарисовать самый нижний из уровней фрактала.

Нарисуйте отрезок, начинающийся в точке `position` и имеющий длину `length` и ориентацию, заданную углом `angle`. Для построения второго конца отрезка воспользуйтесь вспомогательной функцией `angle_to_vector(angle, length)`.

4. Реализуйте метод iteration(), который выполняет одну итерацию. В данном случае необходимо четыре раза вызвать метод `recursive()`. Начальными точками для этих частей будут `position`, `p1`, `p2` и `p3`. Три последние из них можно вычислить, используя функцию `angle_to_vector()` по формулам:

$$p1 = position + angle\_to\_vector(angle, length/3);$$
$$p2 = p1 + angle\_to\_vector(angle - \pi/3, length/3);$$
$$p3 = position + angle\_to\_vector(angle, 2 * length/3);$$

Размер частей в три раза меньше исходного, а ориентация рассчитывается на основании угла angle согласно рисунку 14.

Этап 4. Снежинка Коха

Имея реализацию кривой Коха, сделать снежинку Коха не сложно. Так как класс `KochSnowflake` унаследован от `KochLine`, то реализация базиса рекурсии и рекурсивного шага в нём уже есть. Достаточно только реализовать конструктор и новый вариант метода `base()`.

В методе `base()` трижды вызовите `recursive()` для построения частей фрактала, ориентированных по сторонам равностороннего треугольника, расположенного в центре холста. Размер стороны треугольника можно взять равным $^2/_3$`SIZE`.

Дополнительное задание

Используя эту же схему действий, реализуйте рисование ковра Серпинского с помощью класса `SerpinskiCarpet`. Структура итерации для этого фрактала показана на рисунке

В этом фрактале базовой фигурой является квадрат, выделенный синим цветом на этом рисунке. Итерация будет состоять из рисования базовой фигуры и восьми рекурсивных вызовов.

Установленный в шаблоне размер холста позволяет отобразить ковёр Серпинского до 5-го уровня рекурсии. Рисование более глубоких уровней

будет работать медленно и потребует увеличения размера холста, чтобы результат можно было увидеть.

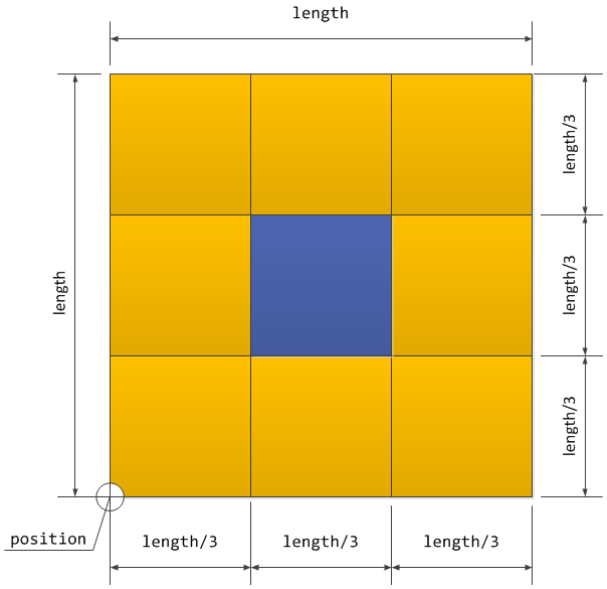


Рис. Структура итерации построения ковра Серпинского

Оценка задания

Методы draw() и recursive() класса Fractal – 20 %.

Метод current_color() класса Fractal – 5 %.

Класс FractalTree – 30 %.

Класс KochLine – 35 %.

Класс KochSnowflake – 10 %.

Класс SerpinskiCarpet – 15 %.

Реализация рисования любого из фракталов задания без использования наследования с помощью рекурсивных функций оценивается в 50% от баллов за реализацию соответствующего класса.

Срок сдачи задания

Для группы, занимающейся во вторник – 13.12.2016.

Для группы, занимающейся в пятницу – 16.12. 2016.

Список литературы

1] Б. Мандельброт, Фрактальная геометрия природы, М.: Институт компьютерных исследований, 2002.

2] P. Prusinkiewicz и A. Lindenmayer, The Algotithmic Beauty Of Plants, New York: Springer-Verlag, 1990, 1996.

3] П. С. Александров, Введение в теорию множеств и общую топологию, М., 1977.