Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Рубежный контроль №1

по курсу «Анализ алгоритмов»

Фракталы

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Оглавление

Введение

В наше время для моделирования нелинейных процессов, таких как течение жидкости в физике, горение пламени, образование облаков, часто используются фракталы - множества, обладающие свойством самоподобия. Также фракталы применяются в области информатики: их используют для сжатия изображений, в компьютерной графики для изображения деревьев, горы и другие ландшафты. Поэтому вопрос быстрого построения фрактальных изображений большой глубины является очень важным. В качестве объекта исследования в данной работе будет рассмотрен фрактал дракона.

1. Аналитическая часть

1.1 Задачи

Цель лабораторной работы: исследовать построение фрактала дракона. Для достижения этой цели поставлены следующие задачи:

- изучить алгоритм построения фрактала дракона;
- реализовать итерационный и рекурсивный варианты алгоритмов;
- оценить эффективность каждой из реализаций по времени и памяти.

1.2 Описание и построение фрактала

Дракон Хартера, также известный как дракон Хартера — Хейтуэя, был впервые исследован физиками NASA — Джоном Хейтуэем, Брюсом Бэнксом, и Вильямом Хартером. Он был описан в 1967 году Мартином Гарднером в колонке «Математические игры» журнала «Scientific American».

Фрактал может быть записан как L-система с параметрами:

- угол равен 90° или $\frac{\pi}{2}$
- начальная строка — FX
- правила преобразования строк:
  $X \mapsto X + YF +$
  $Y \mapsto -FX - Y$

Для построения фрактала дракона берём отрезок, сгибаем его пополам. Затем многократно повторяем итерацию. Если после этого снова разогнуть получившуюся (сложенную) линию так, чтобы все углы были равны 90°, мы получим драконову ломаную. [1]

2. Технологическая часть

2.1 Средства реализации

Для реализации программы был использован язык C++ [3] и графический библиотеки Qt [2]. Для замера процессорного времени была использована функция rdtsc() из библиотеки stdrin.h.

2.2 Реализации алгоритмов

На листингах 3.1 - 3.2 представлены коды реализации алгоритмов построения фрактала.

Листинг 2.1: Итерационная реализация

```
std::vector<Point> get_dragon_fractal(Point p1, Point p2, int n)
{
  std::vector<Point> line;
  line.push_back(p1);
  line.push_back(p2);

  for (int i = 0; i < n; i++)
  {
    int xc = line.back().x(), yc = line.back().y();
    int old_size = line.size();
    for (int j = old_size - 2; j >= 0; j--)
    {
      int xx = xc + (line[j].y() - yc);
      int yy = yc - (line[j].x() - xc);
      line.push_back(Point(xx, yy));
    }
  }

  return line;
}
```

Листинг 2.2: Рекурсивная реализация

```
std::vector<Point> get_dragon_fractal_rec(std::vector<Point> line, int n)
{
  if (n == 0)
    return line;

  std::vector<Point> new_line = get_dragon_fractal_rec(line, n - 1);

  int xc = new_line.back().x(), yc = new_line.back().y();
  int old_size = new_line.size();
```

```
10   for (int j = old_size - 2; j >= 0; j--)
11   {
12     int xx = xc + (new_line[j].y() - yc);
13     int yy = yc - (new_line[j].x() - xc);
14     new_line.push_back(Point(xx, yy));
15   }
16
17   for (auto p : new_line)
18     line.push_back(p);
19   return line;
20 }
```

3. Экспериментальная часть

3.1 Примеры работы

На рисунках 3.1 и 3.2 представлены примеры работы программы, демонстрирующий правильное выполнение алгоритмов.

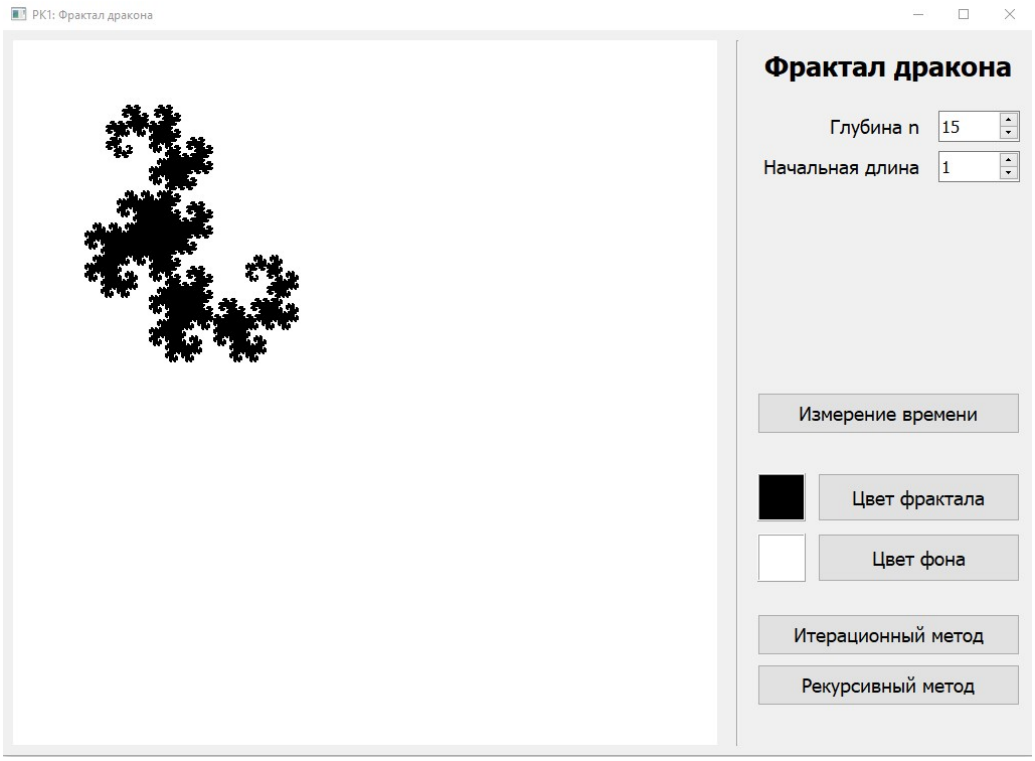

Рис. 3.1: Пример работы программы

3.2 Сравнение работы итерационной и рекурсивной реализации алгоритмов

Для сравнения времени работы двух реализаций алгоритмов построения фрактала глубина построения менялась от 1 до 20. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 3.1 и на рисунках 3.3, 3.4.

Можно сделать вывод о том, что рекурсивный алгоритм проигрывает во времени, причём разница во времени стремительно растёт с увеличением глубины.

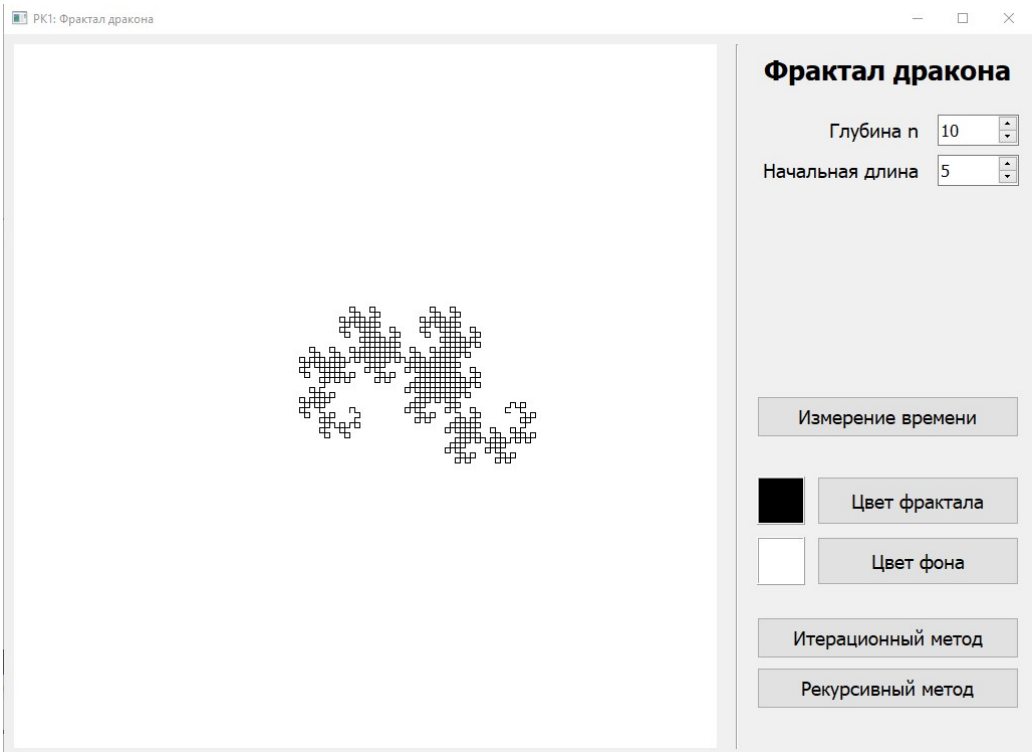

Рис. 3.2: Пример работы программы при увеличенной длины начального отрезка

Таблица 3.1: Время работы реализаций алгоритмов построения фрактала в тактах процессора

Глубина	Итерационный	Рекурсивный
1	4535	9496
2	6614	20311
3	10803	37539
4	13159	59221
5	18511	95390
6	28801	160609
7	43710	277466
8	75896	476731
9	136003	892877
10	232747	1531665
11	426549	2810231
12	1053023	6019168
13	1731680	11664363
14	4072418	22915220
15	9606443	56609665
16	15388502	105251505
17	30714412	205713832
18	57830870	385465827
19	122671112	815684578
20	240363617	1599156703

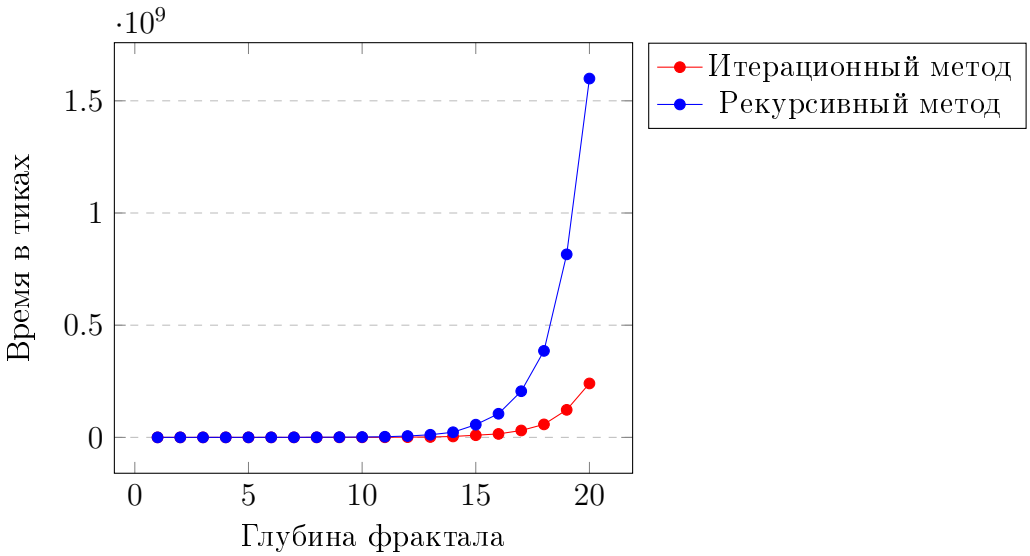

Рис. 3.3: График времени работы реализаций алгоритмов построения фрактала дракона при глубине

Заключение

В ходе выполнения рубежного контроля был изучен метод построения фрактала дракона. Для этого были реализованы итерационная и рекурсивная реализация алгоритма построения. Также была оценена эффективность двух реализаций: итерационный метод построения при большой глубине (больше 16) фрактала значительно выигрывает рекурсивный метод.

Литература

[1] http://mathworld.wolfram.com/DragonCurve.html [Электронный ресурс]

[2] https://doc.qt.io/ [Электронный ресурс]

[3] https://cppreference.com/ [Электронный ресурс]