Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Иркутский государственный университет»
(ФГБОУ ВО «ИГУ»)

Институт математики и информационных технологий

Кафедра вычислительной математики и оптимизации

КУРСОВАЯ РАБОТА

по направлению «Прикладная математика и информатика»

профиль «Математические методы и информационные технологии»

Решение фракталов на языке Haskell

Студента 3 курса очного отделения
группы 02321-ДБ
Округина Никиты Анатольевича

Руководитель:
к. ф.-м. н., доцент
_____ Черкашин Е. А.

Иркутск – 2022

Содержание

Оглавление

ВВЕДЕНИЕ

Разветвления трубочек трахей, листья на деревьях, вены в руке, река, бурлящая и изгибающаяся, рынок ценных бумаг — это все фракталы. От представителей древних цивилизаций до Майкла Джексона, ученые, математики и артисты, как и все остальные обитатели этой планеты, были зачарованы фракталами и применяли из в своей работе. Программисты и специалисты в области компьютерной техники так же без ума от фракталов, так как фракталы бесконечной сложности и красоты могут быть сгенерированы простыми формулами на простых домашних компьютерах. Открытие фракталов было открытием новой эстетики искусства, науки и математики, а также революцией в человеческом восприятии мира. В данной курсовой работе будут изображены фракталы при помощи языка Haskell. Данный язык является гибким для множества математических задач, благодаря обширным количеством библиотек, а так своей производительностью.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Понятие “фрактал”

Понятия фрактал и фрактальная геометрия, появившиеся в конце 70-х, с середины 80-х прочно вошли в обиход математиков и программистов. Слово фрактал образовано от латинского fractus и в переводе означает состоящий из фрагментов. Оно было предложено Бенуа Мандельбротом в 1975 году для обозначения нерегулярных, но самоподобных структур, которыми он занимался. Рождение фрактальной геометрии принято связывать с выходом в 1977 году книги Мандельброта `The Fractal Geometry of Nature. В его работах использованы научные результаты других ученых, работавших в период 1875-1925 годов в той же области (Пуанкаре, Фату, Жюлиа, Кантор, Хаусдорф). Но только в наше время удалось объединить их работы в единую систему.

Роль фракталов в машинной графике сегодня достаточно велика. Они приходят на помощь, например, когда требуется, с помощью нескольких коэффициентов, задать линии

и поверхности очень сложной формы. С точки зрения машинной графики, фрактальная геометрия незаменима при генерации искусственных облаков, гор, поверхности моря. Фактически найден способ легкого представления сложных неевклидовых объектов, образы которых весьма похожи на природные.

1.2 Применение фракталов

Компьютерные системы.

Наиболее полезным использованием фракталов в компьютерной науке является фрактальное сжатие данных. В основе этого вида сжатия лежит тот факт, что реальный мир хорошо описывается фрактальной геометрией. При этом, картинки сжимаются гораздо лучше, чем это делается обычными методами (такими как jpeg или gif). Другое преимущество фрактального сжатия в том, что при увеличении картинки, не наблюдается эффекта пикселизации (увеличения размеров точек до размеров, искажающих изображение). При фрактальном же сжатии, после увеличения, картинка часто выглядит даже лучше, чем до него.

Телекоммуникации.

Для передачи данных на расстояния используются антенны, имеющие фрактальные формы, что сильно уменьшает их размеры и вес.

ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Решетка Серпинского

Это один из фракталов, с которыми экспериментировал Мандельброт, когда разрабатывал концепции фрактальных размерностей и итераций. Треугольники, сформированные соединением средних точек большего треугольника вырезаны из главного треугольника, образовывая треугольник, с большим количеством дырочек. В этом случае инициатор - большой треугольник а шаблон - операция вырезания треугольников, подобных большему. Так же можно получить и трехмерную версию треугольника, используя обыкновенный тетраэдр и вырезая маленькие тетраэдры. Размерность такого фрактала ln3/ln2 = 1.584962501.

2.2 Реализация решетки Серпинского

Разработанная программа рассматривает алгоритм реализации фрактала, представляющего собой решетки Серпинского (Треугольник, в котором другие треугольники), которая в конечном результате выводит данную фигуру на экран в виде 2d рисунка.

Данный код разбит на 2 файла: 1 – в качестве основного файла Main, в котором реализована визуализация фрактала. 2 – в качестве файла реализации функций, на которых основан данный фрактал.

Разберем для начала второй файл, в котором лежит наша основа.

Для начала импортируем нужные для нас библиотеке, в нашем случае это будет библиотека с графическим инструментарием Graphics.UI.GLUT, который имеет всё, что нам нужно для данного кода. А так же уже встроенную функцию mod’ из Data.Fixed.

```
import Graphics.UI.GLUT
import Data.Fixed (mod')
```

Листинг – 2.1

После данных манипуляций приступаем к работе.

Начнем с установки типов данных, с которыми мы будем работать

```
type Point = (GLfloat, GLfloat)
type Area = (GLfloat, GLfloat, GLfloat, GLfloat)
```

Листинг – 2.2

Тип Point – это x и y, который показывает расположение начала рисовки.
Тип Area – это top, right, bottom, left, который показывает область рисовки определенной ветки.

GLfloat – это тип данных, используемый в OpenGL, имеющий тип float.

Дальше у нас идет уже основной код

```
drawFractal :: Area -> Int -> IO()
```

```
drawFractal area 0         = drawTriangle area
drawFractal area iteration = subdivide area iteration

drawTriangle :: Area -> IO()
drawTriangle area = do
   let vertices = toVertices area
       setColorAndDraw vertexPosition = do
          color $ drawingColor vertexPosition
          vertex vertexPosition
   renderPrimitive Triangles $ do
       mapM_ setColorAndDraw vertices



subdivide :: Area -> Int -> IO()
subdivide area iteration = do
   let iteration' = iteration - 1
   mapM_ (\pos -> drawFractal pos iteration') $ subTriangles area



toVertices :: Area -> [Vertex2 GLfloat]
toVertices (top, right, bottom, left) = [topVertex, leftVertex, rightVertex]
   where middleX = (left + right) / 2

         topVertex = Vertex2 middleX top
         leftVertex = Vertex2 left bottom
         rightVertex = Vertex2 right bottom



subTriangles :: Area -> [Area]
subTriangles (top, right, bottom, left) = [topSubTriangle, leftSubTriangle, rightSubTriangle]
   where middleX = (left + right) / 2
         middleY = (top + bottom) / 2
         quarterX = (left + middleX) / 2
         threeQuarterX = (right + middleX) / 2
```

```
        topSubTriangle = (top, threeQuarterX, middleY, quarterX)
        leftSubTriangle = (middleY, middleX, bottom, left)
        rightSubTriangle = (middleY, right, bottom, middleX)



drawingColor :: Vertex2 GLfloat -> Color3
drawingColor (Vertex2 x y) = Color3 red green blue
   where distanceFromCenter = sqrt(x*x + y*y)

        hue = azimuth (x, y)
        saturation = distanceFromCenter
        grayness = 1 - saturation

        secondaryColor = let intensity = realToFrac hue / sixthOfACircle
                         in saturation * realToFrac (1 - (abs ((mod' intensity 2) - 1)))

        (r, g, b) | hue < sixthOfACircle * 1 = (saturation, secondaryColor, 0)
                  | hue < sixthOfACircle * 2 = (secondaryColor, saturation, 0)
                  | hue < sixthOfACircle * 3 = (0, saturation, secondaryColor)
                  | hue < sixthOfACircle * 4 = (0, secondaryColor, saturation)
                  | hue < sixthOfACircle * 5 = (secondaryColor, 0, saturation)
                  | hue < sixthOfACircle * 6 = (saturation, 0, secondaryColor)
                  | otherwise    = (1, 1, 1)

        red = r + grayness
        green = g + grayness
        blue = b + grayness



azimuth :: Point -> GLfloat
azimuth (x, y) | pointIsAtTopRightQuarter    = quarterOfACircle * 0 + acos((y2 + hypotenuse2
- x2) / (2 * (abs y) * hypotenuse))
```

```
            | pointIsAtBottomRightQuarter = quarterOfACircle * 1 + acos((x2 + hypotenuse2 -
y2) / (2 * (abs x) * hypotenuse))
            | pointIsAtBottomLeftQuarter  = quarterOfACircle * 2 + acos((y2 + hypotenuse2 -
x2) / (2 * (abs y) * hypotenuse))
            | pointIsAtTopLeftQuarter     = quarterOfACircle * 3 + acos((x2 + hypotenuse2 - y2)
/ (2 * (abs x) * hypotenuse))
    where quarterOfACircle = pi / 2
          hypotenuse = sqrt(x*x + y*y)

          hypotenuse2 = hypotenuse*hypotenuse
          x2 = x*x
          y2 = y*y

          pointIsAtTopRightQuarter    = x >= 0 && y >  0
          pointIsAtBottomRightQuarter = x >= 0 && y <= 0
          pointIsAtBottomLeftQuarter  = x <  0 && y <= 0
          pointIsAtTopLeftQuarter     = x <  0 && y >  0

sixthOfACircle = pi / 3
```

Листинг – 2.3

Обозначение функций:

- drawFractal – рисовка уже самого фрактала
- drawTriangle – рисовка треугольника
- subdivide – показывает ступени фрактала
- toVertices – определяет вершины
- subTriangles – треугольники, входящие в фрактал
- drawingColor – цвет фрактала по HSV
- azimuth – азимут точек

Теперь начинаем рассматривать наш Main класс, в котором мы просто пишем саму визуализацию нашего фрактала

```
module Main where

import Data.IORef
import Fractals
import Graphics.UI.GLUT
import Control.Concurrent

main :: IO ()
main = do
   initWindow
   initCallbacks
   mainLoop

initWindow :: IO(Window)
initWindow = do
   initialWindowSize $= Size 800 800
   createWindow "Fractal treangle"

initCallbacks :: IO()
initCallbacks = do
   idleCallback $= Just refresh
   displayCallback $= display

refresh :: IO()
refresh = do
   threadDelay 10000

display :: IO()
display = do
   let thirdOfACircle = pi * 2 / 3
       sinThirdOfACircle = sin thirdOfACircle
       cosThirdOfACircle = cos thirdOfACircle

   clear [ColorBuffer]
```

```
drawFractal (1, sinThirdOfACircle, cosThirdOfACircle, -sinThirdOfACircle) 6
flush
```

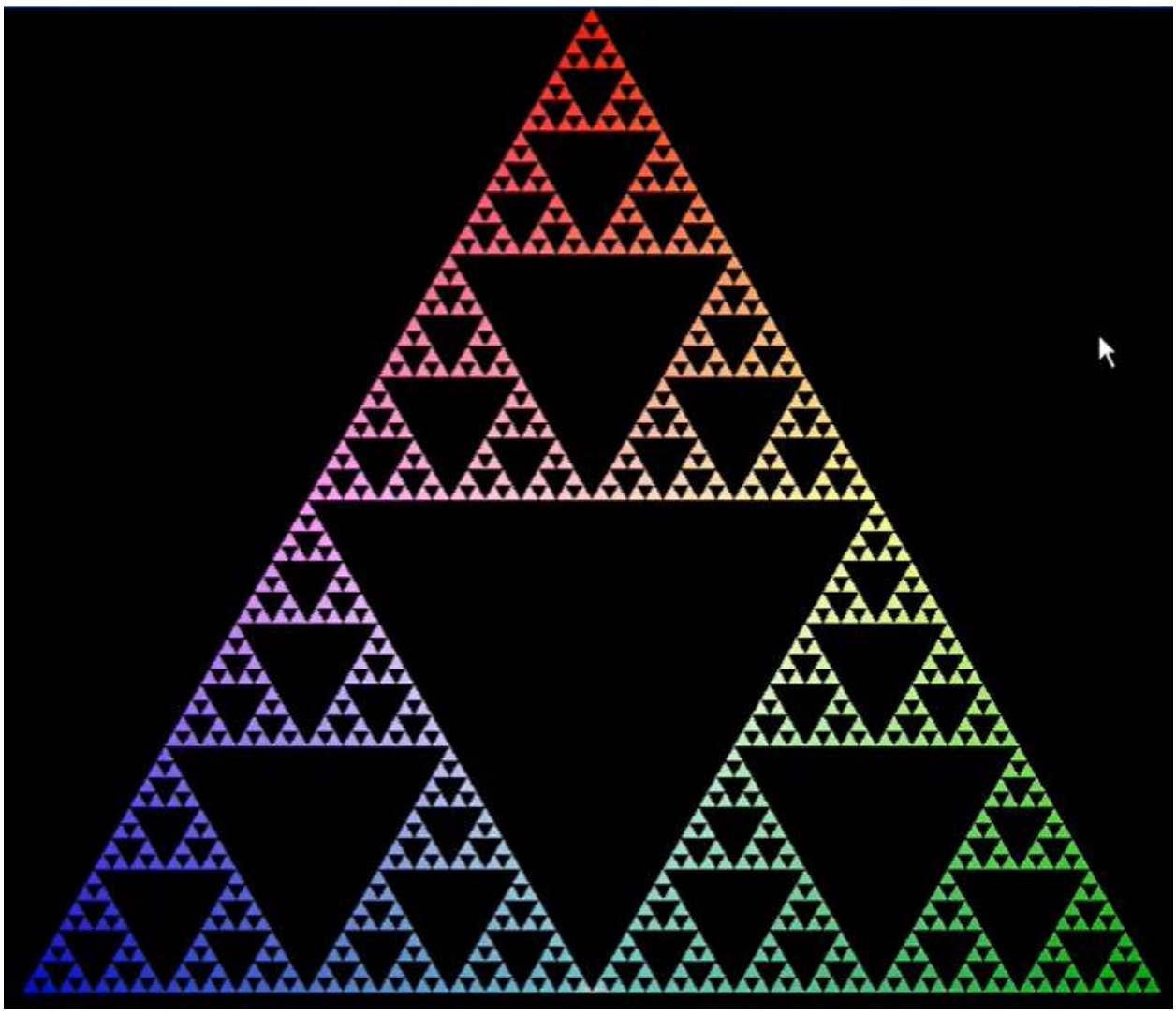
Листинг – 2.4

Обозначение функций:

- initWindow – инициализация окна
- initCallbacks – инициализация обратного вызова
- refresh – обновление окна
- display – отображение нашего фрактала

Так же мы в Main файл импортировали новые встроенные библиотеки, такие как Data.IORef и Control.Concurrent, но один зависим от другого, и всего из них нам нужен threadDeley, который нам дает задержку потока для обновление окна.

Итогом запуска получается данное изображение



Изображение 1. Решетка Серпинского

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разобран объект, как фракталы. Их применение в реальной жизни, тем самым доказана актуальность темы. С помощью функционального языка программирования Haskell была написана программа, реализующая такой известный фрактал, как решетка Серпинского, который наглядным образом обучает использованию рекурсивных методов. Таким образом был рассмотрен функционал языка Haskell, позволяющий реализовать фрактал графически, а также инструментарий, позволяющий проводить дальнейшую работу с ним.

ЛИТЕРАТУРА

1. Миран Липовача Изучай Haskell во имя добра! / Пер. с англ. Леушина Д., Синицына А., Арсанукаева Я.– М.: ДМК Пресс, 2012. – 490 с.: ил. ISBN 978-5-94074-749-9
2. Г.М. Сергиевский, Н.Г. Волченков. Функциональное и логическое программирование. – М.: Академия, 2010. – 320 с.: ил. ISBN: 978-5-7695-6433-8
3. С.В. Микони. Дискретная математика для бакалавра. Множества, отношения, функции, графы. – СПб.: Лань, 2013. – 192 с.: ил. ISBN: 978-5-8114-1386-7
4. Уилл Курт Программируй на Haskell / пер. с англ. Я. О. Касюлевича, А. А. Романовского и С. Д. Степаненко; под ред. В. Н. Брагилевского. – М.: ДМК Пресс, 2019. — 648 с.: ил. ISBN 978-5-97060-694-0